UNIVERSIDAD AUTÓNOMA METROPOLITANA
AZCAPOTZALCO

POSGRADO EN OPTIMIZACIÓN

Metaheurísticas para el problema de ruteo de vehículos con ventanas de tiempo *(VRP-TW)*

Presenta:
Ing. Edwin Montes Orozco

Asesores:
Dr. Roman Anselmo Mora Gutiérrez
Dr. Javier Ramírez Rodríguez

Idónea Comunicación de Resultados
para obtener el grado de

Maestro en Optimización

Ciudad de México, México

2017

Declaración

Yo, Ing. Edwin Montes Orozco, declaro que este trabajo titulado «Metaheurísticas para el problema de ruteo de vehículos con ventanas de tiempo *(VRP-TW)*», es de mi autoría. Yo confirmo que:

- Este trabajo fue realizado en su totalidad para la obtención de grado en esta Universidad.
- Ninguna parte de esta tesis ha sido previamente sometida a un examen de grado o cualquier otra titulación en esta universidad o cualquier otra institución.
- Donde he consultado la obra publicada de los demás, esto se atribuye siempre con claridad.
- Donde he citado del trabajo de los demás, se da siempre la fuente. Con la excepción de estas citas, esta tesis es enteramente mi propio trabajo.

Firma:

Fecha:

«Nadie puede llegar a la cima armado sólo de su talento. Dios da el talento; el trabajo transforma el talento en genio.»

Anna Pavlova

UNIVERSIDAD AUTÓNOMA METROPOLITANA AZCAPOTZALCO

Resumen

Metaheurísticas para el problema de ruteo de vehículos con ventanas de tiempo *(VRP-TW)*

En este trabajo, se presentan 7 técnicas basadas en cuatro metaheurísticas y dos métodos exactos, las cuales son: *Sistema de Hormigas (AS)*, *Búsqueda Armónica (HS)*, *Algoritmo Genético (GA)*, *Búsqueda local iterada (ILS)*, *Algoritmo primal-dual (PDA)* y *Método dual simplex (DSM)* para resolver el *problema de ruteo de vehículos con ventanas de tiempo (VRP-TW)*, haciendo énfasis en los algoritmos híbridos entre estas técnicas, los cuales se denominan *AS-GA*, *AS-HS*, *DSM-AS-PDA*, *AS-ILS*, donde a excepción de *DSM-AS-PDA*, las técnicas involucradas trabajan de manera entrelazada. Con el fin de analizar, comparar y caracterizar el comportamiento de las técnicas desarrolladas, estas se emplearon para resolver 12 instancias del *VRP-TW*.

En *AS-GA*, se utiliza el *GA* para encontrar una mejor solución con la población generada por un ciclo de n hormigas dentro de *AS*, con la cual se actualiza el nivel de la matriz de feromona para el siguiente ciclo de hormigas. Con esto se logra guiar la construcción de una manera más eficaz, y el algoritmo genético ayuda a no converger de manera prematura, sino que debido a la codificación se diversifica el espacio de búsqueda.

Dentro de *AS-HS*, la técnica *HS* sirve para guiar el comportamiento del *AS* a través de los cambios en la matriz de feromona, ya que se aprovecha la información de un número n de ejecuciones de la *HS* guardadas en la memoria armónica (*HM*). En el procedimiento propuesto se retoman las mejores soluciones para la actualización del nivel de la fermona.

Por último, en *DSM-AS-PDA* se utiliza el *optimizador* denominado *Gurobi* ejecutando el método dual simplex para resolver dos relajaciones lineales del problema original. Con las soluciones regresadas por *Gurobi* se inicializa la matríz de nivel de fermona para *AS* y una vez que termina la ejecución de *AS* con base en la mejor solución entregada, se utiliza el algoritmo *PDA* para revisar si la solución encontrada es la óptima. Los resultados muestran que los algoritmos híbridos desarrollados poseen un comportamiento más robusto respecto a las técnicas reportadas en la literatura y a su vez, son capaces de generar mejores soluciones con un número menor de llamadas a la función objetivo en instancias grandes, utilizando menos recursos computacionales.

UNIVERSIDAD AUTÓNOMA METROPOLITANA AZCAPOTZALCO

Abstract

Metaheuristics for the vehicle routing problem with time windows (VRP-TW)

In this work, 7 techniques based on four metaheuristics and two exact methods are presented, which are: *Ant System (AS)*, *Harmonic Search (HS)*, *Genetic Algorithm (GA)*, *Iterated Local Search (ILS)*, *Primal-Dual Algorithm (PDA)* and *Dual Simplex Method (DSM)* to solve the *vehicle routing problem with time windows (VRP-TW)*, emphasizing the hybrid algorithms between these techniques, which are called *AS-GA*, *AS-HS* and *DSM-AS-PDA*; where, with the exception of *DSM-AS-PDA*, these work in an interlaced way. In order to analyze and characterize the behavior of the developed techniques, used a set of test instances for the *VRP-TW*.

In *AS-GA*, the *GA* is used to find a better solution with the population generated by one cycle of n ants within *AS*, with which the level of the pheromone matrix for the next cycle of ants is updated. With this it is possible to guide the construction of a more effective way, and the *GA* helps to avoid the premature converge way, but due to the codification it diversifies the space of search.

Within *AS-HS*, the *HS* technique is used to guide the behavior of *AS* through the changes in the pheromone matrix, since it takes advantage of the information of m executions of the *HS* stored in the harmonic memory (*HM*). In the proposed procedure, the best solutions for the updating of the pheromone level are taken up.

Finally, in *DSM-AS-PDA* is used the *optimizer* called *Gurobi* executing the dual simplex method to solve two linear relaxations of the original problem. With the solutions returned by *Gurobi* the pheromone level for *AS* are initialized and once the execution of *AS* is terminated, based on the best solution delivered, the *PDA* is used to check if this solution is optimal. The results show that the hybrid algorithms developed have a more robust behavior than the techniques reported in the literature and, are able to generate better solutions with a smaller number of calls to the objective function in large instances using less computational resources.

Agradecimientos y dedicatoria

Agradezco y dedico principalmente este trabajo al *CONACyT* por el apoyo económico otorgado y por la oportunidad de realizar mis estudios de maestría.

Al Dr. Roman Anselmo Mora Gutiérrez por el apoyo, las ideas, los conocimientos compartidos, la guía para la realización de esta investigación y por supuesto por su gran amistad.

Al Dr. Javier Ramírez Rodríguez por su dirección, apoyo, motivación e ideas para la realización de esta investigación.

A la Dra. Bibiana Obregón Quintana, por sus puntuales correcciones a este trabajo y su apoyo y motivación para seguir realizando investigación.

A mi esposa Damariz, por estar siempre al pie conmigo, por soportar los ratos de desesperación, por apoyarme cuando las cosas se ponen difíciles, por ser mi motivación, por ser paciente en los momentos en que llega la desesperación a mi persona, por alegrar mis días, por recibirme con un abrazo y una sonrisa cada vez que regreso de la universidad, y sobre todo por el gran amor que siempre me da.

A mis padres Susana y Norberto, porque siempre creyeron en mis sueños, por hacerme creer en mí mismo, por su amor, por ser parte de mi motivación y sus consejos y en especial porque siempre me apoyaron y se que seguirán haciéndolo.

A mi hermano Bryan, ya que al comienzo de mis estudios de maestría se desvelaba conmigo con tal de no dejarme solo estudiando.

A mis abuelos Bertha y Norberto, por sus consejos dados, su crianza y su enseñanza desde los primeros días de mi vida hasta hoy en día.

Al Dr. Eric Rincón García y al Dr. Antonin Ponsich por sus consejos, sus ideas y apoyo durante mis estudios de maestría y en la realización de este trabajo. A la Dra. María Elena Lárraga Ramírez, por aceptar revisar este trabajo y por sus buenos consejos.

Al Posgrado en Optimización de la Universidad Autónoma Metropolitana y al cuerpo académico que lo conforma, por la formación profesional dada.

A la División de Ciencias Básicas e Ingeniería de la Universidad Autónoma Metropolitana Azcapotzalco, por darme el *todo* de mi formación académica.

Y sin restar importancia, a mis amigos y compañeros, Miguel, Gilberto, David y Liz, porque siempre se aprende de las experiencias de cada uno y por seguir conmigo en este sueño llamado *Optimización*.

Índice general

Índice de figuras

Índice de tablas

Lista de abreviaturas

API	**A**pplication **P**rogramming **I**nterface (Interfaz de programación de aplicaciones)
AS	**A**nt **S**ystem (Sistema de hormigas)
AS-GA	**A**nt **S**ystem - **G**enetic **A**lgorithm (Sistema de hormigas-Algoritmo genético)
AS-HS	**A**nt **S**ystem - **H**armonic **S**earch (Sistema de hormigas-Búsqueda armónica)
AS-ILS	**A**nt **S**ystem - **I**terated **L**ocal **S**earch (Sistema de hormigas-Búsqueda local iterada)
BW	**B**and **W**idth (Ancho de banda)
DE	**D**ifferential **E**volution (Evolución diferencial)
DRP	**D**ual **R**estricted **P**rimal (Primal dual restringido)
DSM-AS-PDA	**D**ual **S**implex **A**nt **S**ystem - **P**rimal **D**ual **A**lgorithm (Dual simplex, Sistema de hormigas-Algoritmo primal dual)
DSM	**D**ual **S**implex **M**ethod (Método dual simplex)
GA	**G**enetic **A**lgorithm (Algoritmo genético)
HM	**H**armonic **M**emory (Memoria armónica)
HMCR	**H**armony **M**emory **C**onsidering **R**ate (Razón de exploración)
HMS	**H**armonic **M**emory **S**ize (Tamaño de la memoria armónica)
HS	**H**armonic **S**earch (Búsqueda armónica)
ILS	**I**terated **L**ocal **S**earch (Búsqueda local iterada)
LP	**L**inear **P**rogram (Programa lineal)
LS	**L**ocal **S**earch (Búsqueda local)
PAR	**P**itch **A**djustment **R**atio (Razón de ajuste de tono)
PDA	**P**rimal **D**ual **A**lgorithm (Algoritmo Primal-Dual)
RP	**R**estricted **P**rimal (Primal restringido)
TSP	**T**raveling **S**alesman **P**roblem (Problema del agente viajero)
VRP	**V**ehicle **R**outing **P**roblem (Problema de ruteo de vehículos)
VRP-TW	**V**ehicle **R**outing **P**roblem with **T**ime **W**indows (Problema de ruteo de vehículos con ventanas de tiempo)

Capítulo 1

Introducción

Los problemas de logística y de transporte son tratados como problemas de interés económico, porque se busca generar soluciones que minimicen diversos aspectos tales como la longitud de las rutas, el número de vehículos, los tiempos generados para cada ruta, entre otros. Además, el interés económico, también se debe a la gran cantidad de aplicaciones del mundo real.

En logística existen varios problemas que debido a su complejidad se vuelven de gran interés para la ciencia. Se pueden ver involucradas una serie de disciplinas como la investigación de operaciones, la inteligencia artificial, optimización, entre otras.

Debido a lo anterior, a lo largo de los años se ha planteado una serie de problemas de optimización en logística tales como el problema del agente viajero (*Traveling Salesman Problem, o TSP*) y el problema de ruteo de vehículos *(Vehicle Routing Problem, o VRP)*, los cuales han sido ampliamente estudiados debido a su complejidad [82, 78, 84]. Además,se han propuesto diversas variaciones agregando restricciones, que han dado lugar a nuevos problemas.

En este trabajo se aborda el problema de ruteo de vehículos con ventanas de tiempo (*Vehicle Routing Problem with Time Windows, o VRP-TW*) a través de técnicas metaheurísticas, algoritmos híbridos y una matheurística. El *VRP-TW* implica el cumplimiento de las restricciones temporales (tiempo en el que se sirve a cada cliente), las cuales provocan que las soluciones factibles sean pocas y diversas entre sí. Esto lleva a que la dificultad de encontrar las soluciones óptimas en corto tiempo para una instancia en particular sea aún mayor que en el *VRP* tradicional.

1.- Aplicaciones de los problemas tipo *VRP*

En la Tabla 1.1 se hace notar que el campo de aplicación del tipo de problemas *VRP* es muy amplio, debido a que en la industria el problema de distribución de productos y, en general, los problemas de transporte son de suma importancia para el correcto funcionamiento en los sistemas logísticos.

Área económica	Aplicación
Transporte de materiales	Combustible, gas natural, hormigón.
Salud	Reparto de medicamentos a farmacias.
Banca	Reparto y recolección de dinero en efectivo.
Sector público	Recolección de basura,reparto de correo.
Servicios	Reparación de electrodomésticos a domicilio.
Industria	Suministro de piezas o mercancías entre almacenes.
Educación	Rutas de autobuses escolares.
Defensa	Rutas de aviones espías, logística militar.
Transporte	Planificación de flotas vehículares.

TABLA 1.1: Ejemplos de aplicaciones del *VRP* y sus variantes.

2.- Objetivo General

Diseñar, implementar, probar y caracterizar al menos tres métodos híbridos aplicados al Problema de Ruteo de Vehículos con Ventanas de Tiempo *(VRP-TW)*, cuyo desempeño mejore respecto a métodos reportados en la literatura.

3.- Objetivos Particulares

1. Identificar las características particulares del *VRP-TW*.
2. Revisar el estado del arte para el *VRP-TW* y las técnicas utilizadas para su resolución.
3. Identificar las principales características de los métodos utilizados con mejores resultados.
4. Diseñar varias estrategias que combinen de manera sinérgica los elementos identificados para atacar las problemáticas del *VRP-TW*.
5. Desarrollar métodos híbridos utilizando técnicas exactas y técnicas heurísticas.
6. Realizar el ajuste de parámetros para cada uno de los métodos desarrollados.
7. Probar los métodos desarrollados y analizar si se pueden mejorar.
8. Realizar las pruebas estadísticas y comparaciones de cada método desarrollado y los métodos presentados en la literatura.

La estructura del presente trabajo a fin de lograr los objetivos propuestos, es la siguiente; en el Capítulo 2 se muestran los conceptos principales para la

realización del mismo, el modelo matemático del *VRP-TW*, así como un resumen de la revisión del estado del arte para problemas tipo *VRP*. En el Capítulo 3 se describen las técnicas originales utilizadas, en el Capítulo 4 se presentan las modificaciones realizadas a las técnicas base y los algoritmos híbridos realizados, con el fin de resolver de manera eficiente el *VRP-TW*.

En el Capítulo 5 se muestran los resultados obtenidos con las técnicas implementadas, donde se observa que los algoritmos híbridos *AS-GA*, *AS-HS* y *DSM-AS-PDA* obtienen los mejores resultados, utilizando menos llamadas a la función objetivo y mostrando un comportamiento más robusto respecto a las técnicas que se reportan en la literatura.

Para justificar los resultados numéricos, se realizó el remuestreo de los datos del comportamiento de cada una de las técnicas desarrolladas y la prueba de Wilcoxon para demostrar que los resultados promedio y tiempos de cómputo, son estadísticamente diferentes respecto a las técnicas mostradas en la literatura.

Capítulo 2

Conceptos básicos

1.- Problemas de Ruteo de Vehículos

El problema de distribuir productos desde los depósitos hacia su destino es de gran importancia en muchos sistemas logísticos, por lo cual, en los últimos años se han utilizado ciertas técnicas de optimización como herramientas para una adecuada planificación, ya que esta puede traer considerables ahorros, debido a que se estima que los costos de transporte representan entre un 10 % y 20 % del costo final de los productos [92].

Dada esta forma surge el problema del ruteo de vehículos *(VRP)*, el cual tiene un depósito central que cuenta con una flota de vehículos y un conjunto de clientes que se encuentran geográficamente distribuidos que deben ser atendidos. El objetivo del *VRP* es minimizar el costo requerido para entregar los productos solicitados por un conjunto de clientes mediante la creación de rutas que se originan y terminan en el depósito.

Además, cada cliente es atendido una sola vez y el conjunto de clientes debe ser atendido en su totalidad, para lo cual a cada vehículo se le asigna un sub-conjunto de clientes que debe visitar, satisfaciendo la demanda de estos sin exceder su capacidad.

Los problemas de ruteo de vehículos han sido de gran interés científico debido a que se demostró que pertenecen a la clase de problemas $\mathcal{NP}$-duro[1].[61] y a la gran cantidad de campos de aplicación que estos tienen; como son, transporte, logística, comunicaciones, manufactura, estrategia militar, entre otros [73].

En la Figura 2.1 se muestra una de las formas de representar una solución para el *VRP*, en donde, cada vehículo parte del depósito y regresa al mismo después de visitar a un conjunto de clientes asignados.

[1] $\mathcal{NP}$-duro, es la clase de problemas, tales que, si fuesen polinómicos, se verificaría P=$\mathcal{NP}$

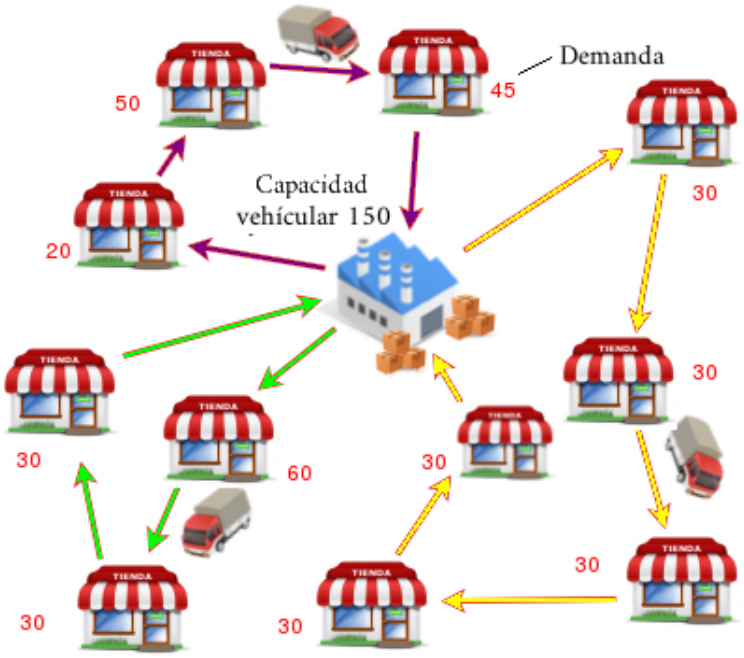

FIGURA 2.1: Ejemplificación de solución para VRP.

2.- Problema de Ruteo de Vehículos con Ventanas de Tiempo

Una forma de extender el *VRP* tradicional consiste en agregar un conjunto de restricciones que permitan asociar una ventana de tiempo a cada cliente, de esta manera se define un intervalo u horario en el que cada cliente debe ser atendido, surgiendo así el *VRP-TW* [84].

En este problema, el tiempo en el que se inicia el servicio a un cliente debe ser mayor o igual al inicio de su ventana de tiempo, y el instante en que se llega a cada cliente debe ser menor o igual al fin de su ventana de tiempo. Si un vehículo llega a la ubicación de un cliente antes del inicio de su ventana de tiempo, debe esperar el horario indicado para servir al cliente.

En la Figura 2.2 se ilustra la forma en que se trabaja una solución para el *VRP-TW* en donde, al igual que en la ejemplificación de solución para *VRP* (Figura 2.1), cada vehículo parte del depósito y visita a un sub-conjunto de clientes asignados satisfaciendo la demanda de estos sin sobrepasar su capacidad, sin embargo, para este tipo de problema también se satisface el límite temporal en el cual el vehículo debe atender a cada cliente.

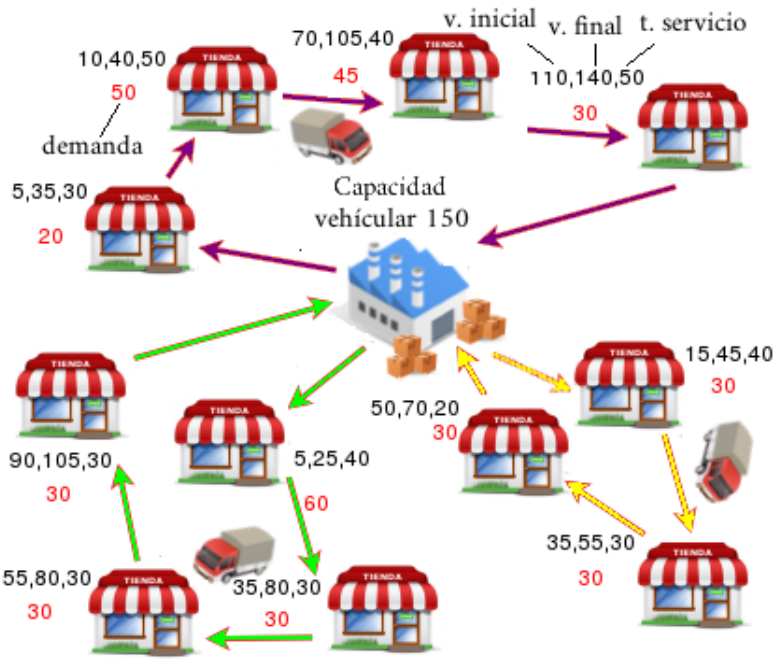

FIGURA 2.2: Ejemplificación de solución para *VRP-TW*.

3.- Modelo Matemático para el *VRP-TW*

En esta sección se presenta el modelo del *VRP-TW*, para lo cual se requiere de la siguiente notación [19]:

$G = (Cl, A)$; gráfica completa no dirigida.
$Cl = \{cl_0, cl_1, cl_2, ..., cl_n\}$; conjunto de nodos (clientes), donde cl_0 es el nodo depósito.
$A = (i, j) : i, j \in Cl, i \neq j$; conjunto de aristas que unen a todos los clientes i y j.
$C = (C_{ij})$; costo de moverse desde el cliente i hacia el cliente j.
$d_i \in \mathbb{Z}^+$; demanda del cliente i.
k; número de vehículos pertenecientes a la flota.
$Q \in \mathbb{Z}^+$; capacidad de los vehículos.

- Se consideran las variables de decisión:

$$X_{ij}^v = \begin{cases} 1, & \text{si el vehículo } v \text{ viaja del cliente } i \text{ al cliente } j; \\ 0, & \text{en otro caso.} \end{cases}$$

- Por último las variables auxiliares:

Y_{iv} = Cantidad entregada por el vehículo v al cliente *i*.
b_i^v = Hora de llegada del vehículo v al cliente *i*.

En esta variante del problema, como se detalla en la sección anterior, además de capacidades sobre los vehículos, cada cliente $cl_i \in Cl \setminus cl_0$ tiene asociada una ventana de tiempo $[e_i, l_i]$ que establece un horario de servicio permitido para que un vehículo arribe a él y un tiempo de servicio o demora s_i (tiempo de descarga o entrega de mercancía).

Por ejemplo, si el camino entre los clientes (i, j) es parte de una solución, b_i y b_j son las horas de arribo a los clientes *i* y *j* respectivamente, entonces las ventanas de tiempo implican que debe cumplirse $b_i \leq l_i$ y $b_j \leq l_j$. Además, se consideran las constantes t_{ij}, que tienen como valor, la distancia existente entre los clientes *i* y *j*.

El modelo de programación matemática para el *VRP-TW* es el siguiente [19]:

Función objetivo:

$$Minimizar \sum_{i=0}^{n} \sum_{j=0}^{n} \sum_{v=1}^{k} C_{ij} X_{ij}^{v} \tag{2.1}$$

Sujeto a:

$$\sum_{i=0}^{n} \sum_{v=1}^{k} X_{ij}^{v} = 1;\ \forall\, j = 1, ..., n \tag{2.2}$$

$$\sum_{j=1}^{n} X_{0j}^{v} = 1;\ \forall\, v = 1, ..., k \tag{2.3}$$

$$\sum_{i=1}^{n} X_{i0}^{v} = 1;\ \forall\, v = 1, ..., k \tag{2.4}$$

$$\sum_{i=0}^{n} X_{ip}^{v} - \sum_{j=0}^{n} X_{pj}^{v} = 0;\ \forall\, p = 0, ..., n;\ v = 1, ..., k \tag{2.5}$$

$$Y_{iv} = d_i \sum_{j=0}^{n} X_{ji}^{v};\ \forall\, i = 1, ..., n;\ v = 1, ..., k \tag{2.6}$$

$$\sum_{i=0}^{n} Y_{iv} \leq Q\ \forall\, v = 1, ..., k \tag{2.7}$$

$$e_i \leq b_i^v \leq l_i\ \forall\, i, j = 0, ..., n;\ v = 1, ..., k \tag{2.8}$$

$$X_{ij}^{v}(b_i^v + s_i + t_{ij} - b_j^v) \leq 0\ \forall\, i = 1, 2, ..., n, v = 1, 2, ..., k \tag{2.9}$$

$$X_{ij}^{v} \in \{0, 1\} \text{ y } Y_{iv} \geq 0;\ \forall\, i = 1, ..., n;\ v = 1, ..., k \tag{2.10}$$

El conjunto de restricciones (2.2) asegura que cada cliente debe ser visitado sólo por un vehículo. Las restricciones (2.3) aseguran que cada vehículo parta solo una vez del depósito, mientras que el conjunto de restricciones (2.4) asegura que cada vehículo regrese al depósito. Las restricciones (2.5) hacen que cada vehículo al arribar a la ubicación de un cliente, salga de esta (restricció de transbordo). El conjunto de restricciones (2.6) impone que la cantidad entregada al cliente i por el vehículo v sea satisfecha en su totalidad. La restricción (2.7) impone que la cantidad entregada en cada ruta no exceda la capacidad del vehículo. La restricción (2.8) asegura que los límites de las ventanas de tiempo son respetados. La restricción (2.9) asegura que cada vehículo v no pueda iniciar el servicio a cierto cliente j, si la suma del tiempo de transporte entre los clientes i y j, la duración del servicio para el cliente i y el tiempo de llegada al cliente i, es mayor que la ventana de tiempo para el cliente j. La restricción (2.10) define el tipo de las variables utilizadas. Finalmente, la función objetivo (2.1) busca minimizar el costo total de las rutas asignadas al conjunto de vehículos.

4.- Estado del Arte

En esta sección se describen los principales avances y técnicas que se han desarrollado desde la década de los años 50 para el *VRP-TW*, donde se muestra la descripción de las principales aportaciones en cada uno de los campos de técnicas de resolución, así como el surgimiento de esta clase de problemas.

- En 1956, Flood desarrolló el primer problema planteado tipo *VRP* (aunque en esa fecha todavía no se conocía el concepto VRP) el cual fue el problema del agente viajero o *TSP* [37].
- El primer trabajo donde se plantea el *VRP* fue presentado por Dantzig y Ramser en 1959, quienes además propusieron una formulación matemática [21].
- En [68] se encuentra la primera referencia del *TSP* múltiple o m-TSP con Miller, Tucker y Zemlin. Este problema es una generalización del TSP en la cual se tiene un depósito y m agentes viajeros.
- En 1964, Clarke y Wright propusieron el primer algoritmo que resultó efectivo para resolver el *VRP*, conocido como el algoritmo de ahorros [15].
- En el año 1967 aparece el *VRP-TW* [78], el cual presenta las variaciones tales como el *VRP-TD* (*VRP* with time deadlines, ventanas con fechas de entrega) en 1986 [9] [88], el *VRP-MTW* (*VRP* con ventanas de tiempo múltiples) el cual fue formulado en 1988 [85]; y *VRP-STW* (*VRP* with soft time windows, ventanas blandas de tiempo) en 1992 [57].
- Además en 1981, Rinooy Kan y Lenstra demostraron que este tipo de problemas se encuentra dentro de la clase $\mathcal{NP}$-duro [61].

En la actualidad, las técnicas para resolver distintas instancias del tipo de problemas *VRP* difieren entre sí dependiendo de la clase de algoritmos a la que pertenecen; es decir, si están basadas en programación lineal, o son técnicas heurísticas clásicas o metaheurísticas [54].

4.1.- Métodos de solución exactos

Estos métodos parten de una formulación como modelos de programación lineal o modelos de programación entera, y llegan a una solución factible entera gracias a los algoritmos de acotamiento del conjunto de soluciones factibles. Por otro lado, se reporta que los métodos exactos son eficientes en problemas de hasta 50 clientes [2] debido a restricciones de tiempo computacional.

En general, los métodos de solución exactos pueden clasificarse principalmente en cuatro grupos:

1. Técnicas de relajación.
2. Búsqueda directa de árbol.
3. Programación dinámica.
4. Programación lineal entera.

Los cuales se describen de la siguiente forma:

- **Técnicas de relajación:** Cuando se trabaja un problema de programación lineal entera, se pueden relajar las restricciones de integralidad, obteniendo una relajación lineal, el cual es un problema de programación lineal continua y por tanto, más sencillo de resolver que el problema original. A continuación, se muestran las mejores técnicas de relajación que se aplican a los problemas del tipo *VRP*:

    1. Relajación lagrangeana. Fue introducida por Held y Karp [48] [49] (1970, 1971), en la implementación de un algoritmo exacto para el *TSP* con la que se obtuvieron resultados prometedores, y posteriormente, ha sido utilizada con éxito en la resolución de otros problemas de optimización combinatoria. Muchos problemas de optimización complicados, son realmente problemas sencillos con un conjunto relativamente pequeño de restricciones complicadas, que alteran su estructura y son las responsables de su gran complejidad. La Relajación lagrangeana consiste en relajar dichas restricciones complicadas y penalizar su violación en la función objetivo a través de pesos determinados, obteniendo una relajación del problema inicial más sencilla de resolver. En [34] y [33] se puede encontrar información detallada acerca de esta técnica.

2. Descomposición de Dantzig-Wolfe. Fue introducida por Dantzig y Wolfe (1960, 1961) [22] [23], se aplica a problemas de programación lineal en las que existe un pequeño grupo de restricciones generales en las que aparecen todas las variables del problema y el resto de restricciones se pueden dividir en grupos que involucran subconjuntos disjuntos de variables.
3. Generación de Planos de Corte. Un plano de corte es una restricción válida para un problema de optimización que se añade a la formulación de una relajación, eliminando la solución óptima de dicha relajación y sin eliminar alguna solución factible del problema original. El objetivo de la introducción de planos de corte es conseguir una sucesión de relajaciones cada vez mejores, con las que se obtienen cotas inferiores próximas a la solución óptima del problema original.

- **Métodos de Búsqueda Directa de Árbol:** En estos métodos, la búsqueda se realiza sobre todos los nodos de un árbol siguendo ciertos criterios específicos propios de cada método [58]. Una descripción breve de las principales técnicas pertenecientes se muestra a continuación.

  1. **Algoritmo de Ramificación y Corte (Branch & Cut):** Propuesto por Gomory en 1958 [44]. Este método resuelve programas lineales sin restricciones enteras utilizando algoritmos simples. Cuando se obtiene una solución óptima que tiene un valor no entero para una variable que ha de ser entera, se utiliza el algoritmo de planos cortantes.
  2. **Algoritmo de Búsqueda de Árbol:** Propuesto por Dakin en 1964 [20]. Para este método, una solución factible se divide en cuatro partes; las aristas que no pertenecen al árbol, las aristas que forman el árbol, las aristas que son incidentes en el primer vértice y las aristas que no son incidentes en el primer vértice. Estos se traducen en restricciones en el modelo matemático y la función objetivo consiste en sumar el costo de todas las aristas en la solución.
  3. **Algoritmo de Ramificación y Acotamiento (Branch & Bound):** Propuesto por Little *et al*. en 1968 [63]. Este método consiste en recorrer cada nodo del árbol desde el nivel superior hasta los nodos hoja, resolviendo en cada uno, un programa lineal y determina qué nodos pueden eliminarse. El algoritmo termina cuando todos los nodos han sido revisados y la solución óptima es la de mayor cota inferior.
  4. **Asignación de cota inferior:** Propuesto por Laporte *et al*. en 1986 [59]. Consiste en asignar una cota inferior que permite disminuir el número de vehículos requeridos para visitar todos los clientes. Esto se realiza por medio del *m-TSP*, como relajación del *VRP*, proporcionando una cota superior para el número de vehículos y transformándolo en un *TSP*.

- **Programación Dinámica:** Este método fue propuesto por Eilon *et al*., en 1971 [31]. En el cual se considera un número fijo de *m* vehículos. Se encuentra primero el costo mínimo alcanzable utilizando un subconjunto *k* de vehículos, teniendo en cuenta la función del costo en la longitud de una ruta, a través de todos los vértices del subconjunto, luego encuentra el costo de todos los subconjuntos de vértices con los *m* vehículos [58].
- **Programación Lineal y Entera:** A continuación se describen las tres técnicas comprendidas dentro de esta clasificación [58]:

  1. Conjunto de particiones y generaciones de columnas: Propuesto por Balinski y Quandt en 1964 [3]. Esta técnica consiste en resolver el problema lineal por medio de la aplicación del Método Simplex; sin embargo, dada la enorme cantidad de columnas (asignaciones factibles) para cada vehículo, en lugar de evaluar todas las columnas respecto a su costo reducido, para determinar cual debe adicionarse a la base, se debe encontrar la columna con el mayor costo reducido.
  2. **Formulación de flujo de vehículos de dos índices:** Propuesto por Fisher y Jaikumar en 1978 [35]. Aquí se introdujo la idea de utilizar las variables binarias x_{ij} para determinar si la arista (*i*, *j*) se utiliza o no en la solución.
  3. **Formulación de flujo de vehículos de tres índices:** Propuesto por Fisher y Jaikumar en 1978 [35]. En las formulaciones basadas en índices, se agregan ya sea 2 o 3 índices al programa lineal original del *VRP*, para así discriminar entre los vehículos (3er índice).
  4. En 2006 Chen *et al*., propusieron un **Programa Lineal Entero-Mixto** para resolver el *VRP-TW* [12].

4.2.- Métodos Heurísticos

4.2.1.- Técnicas Heurísticas.

Las técnicas heurísticas proporcionan soluciones de aceptable calidad mediante una exploración limitada del espacio de búsqueda [71]. Clarke y Wright [15], propusieron el primer método que resultó efectivo para resolver el *VRP*. El desarrollo de las técnicas heurísticas clásicas para resolver el *VRP* se dió entre 1960 y 1990, las cuales se pueden dividir en tres grupos:

1. Métodos constructivos.
2. Métodos de dos fases.
3. Heurísticas de mejora.

Los cuales se describen de la siguiente forma:

- **Métodos constructivos:** Dentro de este grupo se encuentran los algoritmos de ahorros y las heurísticas de inserción, los cuales son ampliamente conocidos dentro del campo de la investigación de operaciones. Los algoritmos de ahorros se desarrollan como se muestra a continuación:

  1. **Algoritmo de Ahorros:** Propuesto por Clarke y Wright en 1964 [15]. En esta técnica, si en una solución se tienen dos rutas diferentes $(0, ..., i, 0)$ y $(0, j, ..., 0)$ se combinan formando una nueva ruta $(0, ..., i, j, ..., 0)$, el ahorro (en distancia) obtenido por dicha unión es definido como $s_{ij} = c_{i0} + c_{0j} - c_{ij}$, por lo que en la nueva solución las aristas $(i, 0)$ y $(0, j)$ no serán utilizadas y se agrega la arista (i, j).
  2. **Mejoras del Algoritmo de Ahorros:** Propuestas por Gaskell en 1967 [39] y Yellow en 1970 [96]. Para evitar algunas problemáticas del algoritmo original, se parte de la relación que a menor distancia entre clientes, mayor ahorro hay en los costos, por lo que se verá reflejado en la solución final. Sin embargo, el algoritmo tiende a formar rutas circulares, para lo cual se introduce un parámetro λ llamado forma de la ruta, el cual evita la formación de rutas circulares mientras se va construyendo una solución.
  3. **Algoritmo de Ahorros basado en acoplamientos:** Propuesto por Desrochers y Verhoog en 1989 [24]. A diferencia del algoritmo original, se decide la union a realizar considerando cómo afecta esta a las posibles uniones en iteraciones siguientes. Para esto, se considera una gráfica que tiene a todas las rutas como nodos y una arista entre dos nodos p y q cuyo peso es el ahorro obtenido si las rutas correspondientes se combinan (siempre que la combinación sea factible). Un acoplamiento de peso máximo sobre dicha gráfica da un conjunto de combinaciones a realizar.
  4. **Heurística de construcción paralela-adaptativa:** Propuesta por Peng en 2011 [75]. Esta técnica es eficiente y eficaz para la construcción de rutas, que es particularmente útil para la generación de las soluciones iniciales para muchas metaheurísticas, para obtener soluciones iniciales de calidad.

  Por otra parte, los trabajos realizados con las heurísticas de inserción se listan de la siguiente forma:

  1. **Heurísticas de inserción secuencial:** Propuesto por Mole y Jameson en 1976 [69]. En esta técnica se utilizan dos medidas para decidir el próximo cliente a insertar en la solución parcial. Por un lado, para cada cliente no visitado se calcula la mejor posición para ubicarlo en la ruta actual teniendo en cuenta solamente las distancias y sin reordenar los nodos que ya están en la ruta. Por lo que es necesario utilizar un incentivo adicional para la inserción de clientes lejanos al depósito.
  2. **Heurísticas de inserción en paralelo:** Propuesto por Christofides, Mingozzi y Toth en 1979 [14]. Esta técnica opera en dos fases. En la

primera fase se determina la cantidad de rutas a utilizar, junto con un cliente para inicializar cada una de las rutas. En la segunda fase se crean dichas rutas y se inserta el resto de los clientes en ellas.

3. **Heurística de inserción guiada:** Propuesta por Yuichi y Olli en 2009 [70]. La técnica primero minimiza el número de vehículos para mantener temporalmente sin servicio a los clientes, lo que permite que la técnica pase por el espacio de soluciones factibles. A continuación, se reduce al mínimo la distancia del recorrido total, utilizando un algoritmo de ascenso a la montaña multi-arranque, permitiendo a la técnica buscar en un vecindario más grande.

- **Métodos de dos fases:** Dentro de estos métodos se encuentran los métodos de asignación elemental, el algoritmo de ramificación y acotamiento truncados, el algoritmo de los pétalos, el método de rutear primero y asignar después y los procedimientos de búsqueda local. A continuación se listan estas técnicas:

  1. Métodos de asignación elemental:
     1.1. **Algoritmo de barrido:** Propuesto por Wren en 1971 [95]. Consiste en formar inicialmente agrupamientos girando una semirrecta con origen en el depósito e incorporando los clientes hasta violar la restricción de capacidad. Una ruta de vehículos es obtenida para el aglomerado resolviendo un *TSP*.
     1.2. **Algoritmo basado en asignación generalizada:** Propuesto por Fisher y Jaikumar en 1981 [36]. Consiste en dos fases, la primera fase consiste en escoger los vértices semilla para construir los agrupamientos. En la segunda fase se asignan los vértices a cada agrupamiento sin violar la capacidad del vehículo resolviendo un *GAP*.
     1.3. **Heurística basada en localización:** Propuesto por Bramel y Simchi-Levi en 1995 [11]. Consiste en establecer las rutas iniciales como un problema de localización con capacidades y los vértices restantes son incluidos en la ruta asignada en una segunda etapa. Para satisfacer las restricciones de capacidad y minimización de los costos, se debe decidir sobre las rutas, que vertices iniciales y terminales conectan cada ruta. Las rutas de los vehículos se construyen entonces, insertando en cada paso, el cliente asignado para que las rutas tengan el menor costo de inserción.
  2. **Algoritmo de ramificación y acotamiento truncados:** Propuesto por Christofides, Mingozzi y Toth en 1979 [14]. En esta técnica el árbol de búsqueda tiene tantos niveles como rutas, y a su vez, cada nivel contiene un conjunto de rutas, para ello se propone una implementación en la cual se asigna una rama en cada nivel y una se descarta.
  3. **Métodos de asignar primero y rutear después:** Propuesto por Fisher y Jaikumar en 1981 [36]. Consiste en dos fases, en la primera se busca generar grupos de clientes que estarán en una misma ruta de la solución final, para esto se busca satisfacer la demanda del

vehículo. Para la segunda etapa, con esos aglomerados se resuelve un *TSP* para cada ruta.

4. **Métodos de rutear primero y asignar después:** Propuesto por Beasley en 1983 [4]. Esta técnica se divide en dos fases. En la primera se calcula una ruta que visita a todos los clientes resolviendo un *TSP*. Luego en la segunda fase, esta ruta se descompones en varias rutas factibles, teniendo en cuenta la solución de la primera fase se determina la mejor partición satisfaciendo la capacidad del vehículo.
5. **Algoritmo de los pétalos:** Propuesto por Ryan, Hjorring y Glover en 1993 [82]. Esta técnica, surge como extensión del algoritmo de barrido y se utiliza para generar diversas rutas llamadas pétalos con el fin de hacer una selección final. Se dispone de un conjunto de rutas, en donde, cada cliente es visitado por varias rutas y se debe seleccionar un subconjunto de las mismas que visite exactamente una vez cada cliente.

- **Heurísticas de mejora:** Mejor conocidos como procedimientos de búsqueda local. Estos métodos se aplican para mejorar una solución ya obtenida. En estos procedimientos se define un conjunto de soluciones vecinas y parte de una solución inicial (generalmente aleatoria), para luego remplazarla por una solución vecina con menor costo. El procedimiento se repite hasta que no pueda mejorar la solución.

  A continuación se listan los principales métodos de búsqueda local:

  1. **El operador de intercambio λ:** Propuesto por Lin en 1965 [62]. Se dice que un intercambio λ consiste en eliminar λ aristas de la solución V y reconectar los λ segmentos restantes. Una solución es óptima si no puede ser mejorada utilizando λ intercambios.
  2. **El algoritmo de Lin-Kernighan:** Propuesto en 1973 [62]. Esta técnica utiliza la idea de intercambiar un subconjunto de arcos por otro, donde dichos subconjuntos (y su cardinalidad) se modifican durante la ejecución del algoritmo. Es decir, se deben determinar dos conjuntos de aristas $x_1, ..., x_k$ e $y_1, ..., y_k$, tales que al realizar un intercambio entre estas disminuya el costo de la solución. Las aristas x deben ser parte de la ruta, ambos conjuntos deben ser disjuntos y, además, eliminar las aristas x y agregar las aristas y formando una ruta cerrada.
  3. **El operador Or-opt:** Propuesto por Or en 1976 [72]. Consiste en eliminar una secuencia de k clientes consecutivos de la ruta y colocarlos en otra posición de la ruta, de modo que permanezcan consecutivos y en el mismo orden.
  4. **GENI y GENIUS:** Propuesto por Gendreau *et al*. en 1992 [41]. Surgen dentro de un método de solución para el *TSP* y tienen como principal característica que la inserción de un cliente en una ruta no necesariamente ocurre entre dos clientes adyacentes.
  5. **Algoritmos de transferencias cíclicas:** Propuesto por Thompson y Psaraftis en 1993 [91]. Son movimientos multi-ruta que intentan eliminar clientes de una ruta y reubicarlos en otra de manera cíclica.

También existe un movimiento que consiste en mover clientes de una ruta a otra en la misma posición.

6. **Operadores de Van Breedam:** Propuestos en 1995 [93]. El primero denominado *Cadena de Traslado (String Relocation)*, se define como una secuencia de *m* nodos que es transferida de una ruta a otra manteniendo el orden en la ruta original. El segundo, denominado *Cadena de Intercambio (String Exchange)*, se intercambia una secuencia de *m* clientes con una secuencia de *n* clientes entre dos rutas.

4.2.2.- Técnicas Metaheurísticas.

Las metaheurísticas se desarrollan a partir de los años 90 y se caracterizan porque realizan un procedimiento de búsqueda para encontrar soluciones de aceptable calidad. Estas técnicas están diseñadas para resolver problemas difíciles de optimización, en los que las heurísticas clásicas no son efectivas ni eficientes. Dentro de estas, se encuentran el recocido simulado, redes neuronales, búsqueda tabú, algoritmos genéticos, algoritmos de hormigas y búsqueda de vecindades variables, etc.

A continuación se muestra el listado de las principales metaheurísticas aplicadas al *VRP*:

1. **Búsqueda en vecindades variables:** Propuesta por Hansen y Mladenovic en 1974 [47]. En esta técnica, se realiza una búsqueda local, en la cual, se lleva a cabo distintos cambios de vecindarios; aquí, generalmente se realizan vecindarios con cambios pequeños al inicio y conforme al paso de la búsqueda, se utilizan vecindarios con cambios más fuertes o visceversa. Por otro lado, para evitar que la búsqueda se vuelva aleatoria es recomendable utilizar distintos tamaños de vecindario al inicio de la búsqueda y cuando esta se encuentra avanzada.

2. **Algoritmos genéticos:** Propuestos por Holland en 1975 [51]. Esta técnica está inspirada en la teoría de la evolución darwiniana, se parte de una población inicial de individuos que representan soluciones iniciales. Seguidamente, la población evoluciona mediante la aplicación de operadores evolutivos que combinan y modifican a los individuos de la población creando una nueva. Habitualmente se trabajan tres operadores genéticos: selección, cruza y mutación.

3. **Recocido simulado:** Propuesto por Kirkpatrichk *et al.*, en 1983 [56]. Esta metaheurística está inspirada en el proceso de recocido del acero y cerámicas, una técnica que consiste en calentar y luego enfriar lentamente el material para variar sus propiedades físicas. El calor causa que los átomos aumenten su energía y que puedan así desplazarse de sus posiciones iniciales; el enfriamiento lento les da mayores probabilidades de recristalizar en configuraciones con menor energía que la inicial (óptimo local).

   3.1. **Recocido simulado determinístico:** Propuesto por Dueck y Scheurer en 1990 [29]. A diferencia del Recocido Simulado original, se propone una actualización determinista que acepta reordenamientos que conducen a soluciones de un mayor costo, y así ampliar el espacio de busqueda.

4. **Redes neuronales:** Propuestos por Hopfield y Tank en 1985 [53]. Se definen como un paradigma de auto-aprendizaje y procesamiento automático inspirado en la forma en que funciona el sistema nervioso. Se trata de un sistema de interconexión de neuronas que colaboran entre sí para producir un estímulo de salida (en este caso soluciones).

5. **Búsqueda tabú:** Propuesta por Glover en 1986 [42]. Consiste en realizar una búsqueda local aceptando soluciones que mejoran el comportamiento del costo de tal manera que en cada iteración se desplaza de una solución (s_t) a otra mejor (s_{t+1}) dentro de un subconjunto de soluciones cercanas. Como s_{t+1} no necesariamente es el menor costo, se utiliza una memoria de corto plazo que registre algunos atributos de soluciones ya visitadas.

   5.1. **Procedimiento de memoria adaptativa:** Propuesta por Rochat y Taillard en 1994 [81]. Estas estrategias son favorables para encontrar soluciones a los problemas debido a que desarrollan una búsqueda inteligente, basada en el empleo de estructuras de memoria.

6. **Algoritmos de hormigas:** Popuestos por Dorigo *et al.*, en 1992 [25]. Esta técnica, está inspirada en la forma en que las hormigas se mueven para encontrar alimento. Cada hormiga, al recorrer un camino deja un rastro de feromona, el cual guiará a otras hormigas a seguirlo; mientras más hormigas recorran este camino, se fortalecerá con su feromona. Entonces, la feromona se acumulará mayormente en los caminos cortos.

En 2005 Cho y Wan [13] presentan un metaheurística que se basa en aceptar y combinar con el vecino más próximo, y procedimientos de cambio para resolver el *VRP-TW*. En ese año, Bouthillier y Crainic [10] proponen una metaheurística paralela, basada en re-alojamiento de los depósitos.

Por otro lado, Potvin *et al*. [52] proponen un híbrido de dos fases, donde el objetivo de la primera fase es la minimización del número de vehículos, mientras que el objetivo de la segunda fase es minimizar la distancia total mediante búsqueda tabú. En ese mismo año, una metaheurística eficaz, fue propuesta, la cual consiste en una evolución activa guiada para el *VRP-TW*. Por último, en el año 2011 en [38] propusieron un algoritmo evolutivo multi-objetivo para el *VRP-TW*.

4.2.3.- Algoritmos híbridos y de combinación.

Se crean tomando ideas y métodos de los algoritmos exactos y las técnicas heurísticas. Dumitrescu y Stützle [30] describen que existen hibridaciones centrándose en la búsqueda local que son reforzadas por el uso de algoritmos exactos. Puchinger y Raidl [77] definen dos principales ramas de algoritmos híbridos, las cuales se definen de la siguiente forma:

1. Combinaciones colaborativas: Significa que los algoritmos intercambian información, pero no son parte de los otros. Los métodos exactos y heurísticos pueden ser ejecutados de forma secuencial, entrelazados o en paralelo.

2. Combinaciones integradoras: Por integración se entiende que una técnica es un componente incrustado subordinado de otra. Por lo tanto, hay una distinción de la técnica que trabaja como principal, que puede ser tanto exacta como heurística, y al menos un receptor integrado.

Durante los últimos años, numerosos estudios han sido realizados para generar buenas soluciones para el *VRP-TW* con algoritmos genéticos híbridos con diferentes heurísticas de construcción [8, 6], búsquedas locales [54, 90] y otras meta-heurísticas tales como búsqueda tabú [50] y algoritmos de hormigas [5].

Los principales algoritmos híbridos propuestos en los últimos años se presentan a continuación:

1. **Algoritmo genético híbrido generacional**. Propuesto por Khanh *et al.*, en 2014 [55]. En esta técnica se utiliza un algoritmo genético, en el cual se realiza una búsqueda local para el proceso de cruza a partir de las soluciones generadas hasta el momento por el algoritmo, es decir, la búsqueda local se aplica con las soluciones que se tienen al momento y se lleva a cabo utilizando distintos tipos de vecindarios para llegar a óptimos locales, los cuales son regresados como individuos para la siguiente generación.

2. **Búsqueda local iterada con búsqueda tabú granular**. Propuesta por Escobar *et al.*, en 2014 [32]. Se lleva a cabo el proceso de búsqueda local, y en el procedimiento de perturbación, se aplica la búsqueda tabú granular cuando la mejor solución encontrada no se puede mejorar por un número determinado de iteraciones.

3. **Algoritmos híbridos celulares hibridado con metaheurísticas**. Propuesto por Willmer *et al.*, en 2014 [94]. En esta técnica se mezclan distintas metaheurísticas basadas en construcción con el algoritmo genético, es decir, el proceso de generación de la siguiente población se lleva de forma paralela aplicando distintas metaheurísticas a los individuos actuales.

4. **Algoritmo híbrido con reencadenamiento de trayectorias y GRASP**. Propuesto por Zhou *et al.*, en 2013 [97]. La metodología *GRASP (Búsqueda*

Adaptativa Glotona Aleatorizada) y reencadenamiento de trayectorias se llevan de forma secuencial, esto es, se realiza el GRASP de forma normal, y las mejores soluciones obtenidas al final son unidas por medio del reencadenamiento de trayectorias, por lo que se obtienen mejores soluciones intermedias entre estas.

5. **Algoritmo genético híbrido con búsqueda glotona**. Propuesto por Soonpracha *et al.*, en 2015 [86]. Consiste en una modificación de un algoritmo genético y una adaptación de búsqueda glotona hibridizada con métodos de búsqueda local intra-ruta.

Capítulo 3

Técnicas de resolución base

1.- Descripción de las técnicas

En esta sección, se muestran las técnicas base utilizadas en este trabajo. Se listan sus propiedades originales descritas en la literatura. Para este trabajo, se realizaron diversas modificaciones en las técnicas, con el objeto de aprovechar la estructura del problema y obtener mejores resultados; por lo cual, en el Capítulo 4 se da la descripción de estas técnicas y sus modificaciones para resolver el *VRP-TW*.

1.1.- Sistema de Hormigas (AS, por sus siglas en inglés)

Es una metaheurística propuesta por Dorigo *et al.* [25], que se inspira en una metáfora natural, principalmente en la comunicación que existe entre las hormigas que les permite encontrar caminos desde su nido hasta las fuentes de alimento. La comunicación entre las hormigas, se basa en la segregación de sustancias químicas denominadas *feromonas*. Por ejemplo, si se tiene una hormiga que comienza su recorrido, al detectar un camino de feromonas con cierta probabilidad esta hormiga lo seguirá y lo fortalecerá con la suya. Por lo tanto, la probabilidad de que otras hormigas sigan una trayectoria dada, aumenta con el número de hormigas que anteriormente lo han seguido. Esto conduce a la aparición de caminos más cortos, ya que por estos pasarán una mayor cantidad de hormigas, y existirá mayor concentración de feromona [25] [26].

En el procedimiento de esta técnica, un número de hormigas construyen soluciones de manera "aleatoria" y glotona en cada ciclo. Cada hormiga elige el siguiente elemento a incorporar en su solución parcial actual basada en algunas evaluaciones de ese elemento (probabilidad de elección), que va dada por la cantidad de feromona y por un peso asociado (generalmente la distancia al mismo).

Existen tres tipos de *AS* (densidad hormiga, cantidad hormiga y ciclo hormiga) [27], los cuales se diferencian por el momento en el cual se actualiza el nivel de feromona. En el presente trabajo se utiliza la version de *ciclo hormiga*,

el cual utiliza la actualización del nivel de feromona al terminar un ciclo de construcción de soluciones para n hormigas.

1.2.- Búsqueda Local Iterada (ILS, por sus siglas en inglés)

Búsqueda Local Iterada es una metaheurística propuesta por Lourenço, *et al.* [67] [66], y trabaja con la idea de realizar de manera iterativa diversos cambios de vecindario dentro de una búsqueda local. El procedimiento de la técnica se puede ver de la siguiente forma: dada una solución obtenida por la búsqueda local se aplica una perturbación que da lugar a una solución intermedia. Al aplicar nuevamente la búsqueda local a esta solución, se obtiene una nueva solución que, si supera un criterio de aceptación, pasa a ser la nueva solución alterada. En la actualidad, se ha propuesto aplicar cualquier otra metaheurística en lugar de la búsqueda local guiada por la misma idea. A continuación, se muestran algunas características de la *ILS*.

- Debe hacerse una buena elección de los vecindarios.
- No siempre la mejor búsqueda local lleva a una mejora en *ILS*.
- Si el tiempo de computación es fijo, puede ser mejor aplicar con más frecuencia un algoritmo de búsqueda local más rápido aunque menos efectivo, que uno más lento y más poderoso.

Para comprender mejor el funcionamiento de la búsqueda local iterada, es necesario entender cómo funciona la búsqueda local, el criterio de aceptación y las perturbaciones, por lo cual en las siguientes subsecciones se detalla cada uno de estos elementos.

1.2.1.-Búsqueda Local (LS, por sus siglas en inglés)

En esta técnica, se genera una solución aleatoria dentro del espacio de búsqueda que se evalúa mediante la función objetivo, después, esta solución inicial se modifica utilizando cierta estructura de vecindario y se evalúa del mismo modo. Si esta nueva solución es mejor que la actual, es sustituida, de lo contrario permanece sin cambios, esto se realiza hasta que se cumpla con el criterio de paro (generalmente cierto número de iteraciones). El Pseudocódigo 1 ilustra el procedimiento general de la *LS*.

Pseudocódigo 1 Búsqueda Local

1: Generar una solución aleatoria S
2: Evaluar el valor de F.O. para S
3: **mientras** no se cumpla el criterio de paro **hacer**
4: Mover en cierto vecindario la solución S generando a S'
5: Evaluar el valor de F.O. para S'
6: **si** $F(S') \leq F(S)$ **entonces**
7: $S = S'$
8: **fin si**
9: **fin mientras**
10: **devolver** la solución S

1.2.2.- Perturbaciones

Para el correcto funcionamiento de esta técnica, se recomienda que las perturbaciones tomen en cuenta algunos aspectos del problema que se atacará, ya que una buena perturbación ayuda a encontrar mejores soluciones dentro del espacio de búsqueda, de lo contrario, si se tiene una perturbación demasiado fuerte, se puede tener una búsqueda que cada vez que perturbe una solución obtenga una solución igual a una generada de forma aleatoria y la probabilidad de encontrar mejores soluciones será baja.

1.2.3.- Criterio de Aceptación

El criterio de aceptación se utiliza para decidir en qué momento la solución perturbada y mejorada por la búsqueda local es aceptada; este parámetro es de gran importancia debido a que debe permitir que la intensificación y la diversificación dentro de la búsqueda se encuentren equilibrados. En este trabajo se utiliza el criterio de aceptar la solución perturbada si esta es mejor que la solución actual.

En el Pseudocódigo 2 se muestra el esquema básico que gobierna la búsqueda local iterada [67, 66]:

Pseudocódigo 2 Búsqueda Local Iterada

1: Generar una solución aleatoria S
2: Aplicar un algoritmo de búsqueda local, que proporcione un óptimo local S^* utilizando el Pseudocódigo 1.
3: **mientras** no se cumpla el criterio de parada **hacer**
4: Aplicar una perturbación a la solución S^* para transformarla en S'
5: Emplear el algoritmo de búsqueda local para obtener $S^{*'}$
6: Si $S^{*'}$ cumple un criterio de aceptación, considerar a $S^{*'}$ como el siguiente $S*$
7: **fin mientras**
8: **devolver** la solución S

1.3.- Búsqueda Armónica (HS, por su siglas en inglés)

Búsqueda armónica fue propuesta por Geem [40], y es una metaheurística inspirada en la manera en que los músicos buscan la mejor armonía dentro de la composición musical. Esta técnica ha sido empleada para resolver problemas de optimización complejos de manera existosa. Dentro de *HS*, una solución es similar a una armonía, mientras que sus operadores simulan el proceso de improvisación.

En comparación con otras metaheurísticas, para *HS* el tiempo de ejecución por lo general es pequeño [60, 64]. Los parámetros de control para la metaheurística *HS* son el tamaño de la memoria armónica *(HMS)* que define el número de elementos que son almacenados durante el proceso de optimización, la razón de exploración o probabilidad de construir una nueva solución a partir de los elementos previamente almacenados en la memoria *HM* denominado *(HMCR)*, la razón de ajuste de tono o probabilidad de volver a modificar una armonía nombrado *(PAR)*, el ancho de desplazamiento o la fuerza de la perturbación aplicada a la solución denominado *(BW)* y el número de improvisaciones *(NI)* que representa el número de iteraciones dadas a la técnica.

1.4.- Algoritmo Genético (GA, por sus siglas en inglés)

Es una técnica propuesta por Holland [51] y se fundamenta en una metáfora natural, básicamente, imita la forma en que los individuos de una población evolucionan y se adaptan a su entorno de acuerdo con el principio de la selección natural darwiniana. Bajo este paradigma, una población de soluciones (conocida como población de cromosomas o individuos) evoluciona de una generación a la siguiente a través de la aplicación de los operadores genéticos como cruza y mutación.

Mediante el proceso de selección, las soluciones con mejor aptitud pasan a la siguiente generación, mientras que en procedimiento de cruza se toman dos soluciones padres y se combinan las características más deseables de cada uno para crear una o dos soluciones hijo. Esto se repite hasta que se obtiene una nueva población de descendientes. Finalmente cada descendiente se perturba al azar por un operador de mutación. A partir de la población inicial, este ciclo se repite para una serie de generaciones, y la mejor solución encontrada se devuelve al final.

1.5.- Algoritmo primal-dual (PDA, por sus siglas en inglés)

Esta técnica es aplicada a programas lineales y problemas de optimización que pueden ser resueltos en tiempo polinomial. Consideremos el programa lineal

$$Min\ c^T x \tag{3.1}$$

sujeto a:

$$Ax \geq b$$
$$x \geq 0$$

Y su dual:

$$Max\ b^T y \tag{3.2}$$

sujeto a:

$$A^T y \leq c$$
$$y \geq 0$$

Donde $A \in Q^{m \times n}, c, x \in Q^n, b, y \in Q^m, y^T$ denota el vector transpuesto. Para facilitar la representación se supone que $c \geq 0$. En el método primal-dual, se supone que el primal tiene una solución factible y al dual; inicialmente se puede establecer $y = 0$ [74]. En el método primal-dual, se puede encontrar una solución x al primal que cumpla las condiciones de holguras complementarias con respecto a y, demostrando así que tanto x como y son óptimas, o bien se puede encontrar una nueva solución factible dual con un mayor valor de la función objetivo [43].

Se debe considerar lo que significa que x sea complementario a y; sea A_i la i-ésima fila de A y A^j la j-ésima columna de A; para los programas lineal 3.1 y dual 3.2, existen dos tipos de condiciones para holguras complementarias. El primero correspondiente a las variables primales, es decir

$$x_j > 0 \Rightarrow A^j y = c_j \tag{3.3}$$

Siendo $J = \{j | A^j y = c_j\}$.

En segundo lugar, correspondientes a las variables duales, es decir:

$$y_i > 0 \Rightarrow A_i x = b_i \tag{3.4}$$

Siendo $I = \{i | y_i = 0\}$.

El conjunto de condiciones 3.3, 3.4 son satisfechas si x es factible en 3.1. Ahora, se debe suponer que se tiene una y factible en 3.2 si se pudiera encontrar de cualquier manera una x que satisfaga

$$x_j = 0$$

Entonces, x y y serían óptimas. Entonces el algoritmo primal-dual se deriva de la idea de buscar una x, dada una y. Se busca esa x resolviendo un problema auxiliar llamado primal restringido (RP), determinado por el primal con el que se está trabajando. Si la búsqueda de x no tiene éxito, nunca se obtendrá la información del dual de *RP* denominado *DRP*, que dice cómo mejorar la solución y con la que se inició; de esta manera, se converge a la optimalidad en un número finito de pasos [74].

A continuación, se describe el procedimiento general del algoritmo.

1.5.1.- Procedimiento general

El algoritmo primal-dual comienza con una solución y factible en el programa dual 3.2. En el caso que $c \geq 0$, se puede tomar $y = 0$ inmediatamente como un punto inicial factible en el dual. Cuando c es no negativo, se puede encontrar una solución y fácilmente introduciendo la variable X_{n+1} al problema primal 3.1 y la restricción

$$x_1 + x_2 + ... + x_n + x_{n+1} = b_{m+1} \tag{3.5}$$

Donde b_{m+1} es mayor que la suma de cualquier valor de la posible solución $x_1, ..., x_n$ a 3.1 y el costo $C_{n+1} = 0$. Claramente la ecuación 3.5 no cambia la solución a 3.1. El dual del nuevo primal se obtiene agregando una nueva variable y_{n+1} como se muestra [74]:

$$max\ w = y'b + y_{m+1}b_{m+1}$$

$$y'A_j + y_{m+1} = C_j; j = 1, ..., n$$

$$y_{m+1} \leq 0$$

Una solución factible a este *LP* es simple:

$$y_i = 0; i = 1, ..., m$$

$$y_{m+1} = min\{c_j\} < 0$$

$$1 \leq j \leq m$$

La última desigualdad se deduce de la suposición de que c es negativo. En cada iteración debe mantenerse la factibilidad de la solución y.

Ahora, se tiene una y factible en el programa dual 3.2, y algunas de las restricciones de desigualdad

$$y'A_j \leq c_j$$

Se cumplirán estrictamente y otras no. Se define el conjunto de índices J por

$$J = \{j_y^{'} :\ A_j = c_j\} \tag{3.6}$$

Sabemos de 3.3 y 3.4 que una x factible en P es óptima exactamente cuando

$$x_j = 0 \text{ para todo } j \notin J \tag{3.7}$$

Esto equivale a buscar una x que satisfaga

$$\begin{gathered} \sum_{j \in J} a_{ij} x_j = b_i;\ i = 1, ..., m \\ x_j \geq 0,\ j \in J \\ \sum_{j \in J} a_{ij} x_j = b_i;\ i = 1, ..., m \end{gathered} \tag{3.8}$$

Este conjunto de igualdades utiliza sólo las columnas de A correspondientes a las igualdades en D, que a su vez corresponden al conjunto J; Por esta razón se denomina J al conjunto de columnas admisibles. Para buscar tal x, se crea un nuevo LP, llamado el primal restringido, como se muestra a continuación [74]:

$$\begin{gathered} min\ \varepsilon = \sum_{i=1}^{m} x_i^a \\ \sum_{j \in J} a_{ij} x_j + x_i^a = b_i;\ i = 1, ..., m \\ x_j \geq 0,\ j \in J \\ x_j = 0,\ j \in J \\ x_i^a \geq 0 \end{gathered} \tag{3.9}$$

Es decir, se introducen variables artificiales, una para cada ecuación en P (Función 3.1). El primal restringido 3.9 puede ser resuelto usando el algoritmo simplex ordinario. Si la solución óptima a RP es $\varepsilon_{opt} = 0$, entonces se tiene una solución a 3.8 y por lo tanto una solución óptima a P. Lo que es de interés ahora, es lo que sucede cuando $\varepsilon_{opt} = 0$ en RP, por lo que se considera el dual de RP (DRP) [74]

$$max\ w = y'b \tag{3.10}$$

$$y'A_j \leq 0; j \in J \tag{3.11}$$

$$y_i \leq 1; i = 1, ..., m \tag{3.12}$$

$$y_i \neq 0 \tag{3.13}$$

Se denota la solución óptima de DRP, obtenida cuando RP se resuelve, por y'. Ahora se ha tratado de encontrar una x factible utilizando sólo columnas admisibles, pero si $\varepsilon_{opt} > 0$ en RP, hay que elegir otra solución y.

En la Figura 3.1 se muestra la forma en que trabaja el algoritmo primal-dual utilizando los diversos programas lineales.

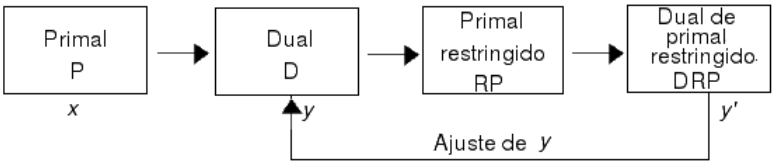

FIGURA 3.1: Procedimiento algoritmo primal-dual [74].

2.- Herramientas

2.1.- Lenguaje de programación

Durante el desarrollo de las técnicas descritas anteriormente, se utilizó el lenguaje de programación *C*. Esto es, debido a que la mayoría de las implementaciones que se tienen reportadas en la literatura se encuentran en este lenguaje de medio/bajo nivel. Por otro lado, las bibliotecas y API's *(Application Programming Interface)* para trabajar con otras herramientas son amplias y robustas.

2.2.- Gurobi [46]

Debido a que las implementaciones de las técnicas descritas anteriormente son puramente en lenguaje *C*; para la técnica *DSM-AS-PDA* se utiliza la herramienta *Gurobi*[46] haciendo uso específico de la *API* para *C* disponible para esta.

Una vez que se tienen instalados e inicializados tanto el optimizador como la *API*, es necesario crear un entorno de trabajo. El entorno actúa como un contenedor para todos los datos asociados con un conjunto de corridas. Por lo general, sólo se necesita un entorno en el programa, incluso si se desea trabajar con múltiples modelos de optimización. Una vez que se haya terminado de utilizar dicho entorno se deben liberar los recursos asociados.

Se puede crear uno o más modelos dentro de un entorno. Un modelo consta de un conjunto de variables, una función objetivo lineal o cuadrática respecto a estas, y un conjunto de restricciones. Cada variable tiene asociado un límite inferior y un límite superior, además pueden ser del tipo continuo, binario, entero, semi-continuo o semi-entero, así como los coeficientes del objetivo. Cada restricción lineal tiene un sentido asociado (menor que o igual, mayor que o igual, o igual), y el valor de lado derecho.

Un modelo se puede especificar en conjunto, o puede ser construido de forma incremental añadiendo variables una a una y después las restricciones. Los modelos son entidades dinámicas; donde siempre se puede añadir o eliminar variables y/o restricciones. Cada una de las variables y restricciones se hacen

referencia a lo largo de la interfaz *Gurobi C* utilizando sus índices. Los índices de variable/restricción se añaden al modelo.

En las siguientes subsecciones se muestra el manejo de las técnicas utilizadas para resolver el modelo dependiendo del tipo de modelo utilizado en la herramienta, además del formato para generar los modelos lineales para *Gurobi*.

2.2.1.- Manejo de los diferentes tipos de problemas

A menudo nos referimos a un modelo de optimización como un programa lineal (PL) donde la función objetivo, y las restricciones son lineales, y las variables continuas. Si el objetivo es cuadrático, el modelo es un programa cuadrático (QP). Si ninguna de las restricciones es cuadrática, el modelo es un programa cuadrático-restringido (QCP). Por otro lado, si el modelo contiene algunas variables enteras, variables semi-continuas, variables semi-enteras, o restricciones especiales, el modelo es un programa entero mixto (MIP). Existen casos especiales de MIP, incluyendo programa lineal entero mixto (MILP), programa cuadrático entero mixto (MIQP), programa entero mixto de forma cuadrática con restricciones (MIQCP) [46].

Una vez que se haya construido el modelo, se puede calcular una solución. Por defecto, para programas lineales se puede utilizar una optimización simultanea (resuelve el programa lineal con distintas estrategias). Los algoritmos de *Gurobi* dan un seguimiento cuidadoso al estado del modelo, por lo que sólo realizará una optimización adicional si los datos relevantes han cambiado desde la última vez que el modelo fue optimizado[1]. Por otro lado, se puede descartar información de soluciones previamente calculadas y se reinicia la optimización sin necesidad de cambiar el modelo .

Si se tiene que un modelo resulta infactible, se cuenta con algunas opciones para tratar de diagnosticar la causa de la infactibilidad y repararla. Para esto se puede calcular una relajación de factibilidad para el modelo. Esta relajación permite encontrar una solución que minimiza la magnitud de la violación de las restricciones. En el Apéndice A se muestran algunas de las funciones principales de la interfaz *Gurobi C*.

2.2.2.- Formato LP de Gurobi

El formato *LP* facilita la captura de un modelo de optimización. Una limitación del formato LP es que no conserva varias propiedades del modelo. En particular, los archivos LP no conservan orden de las columnas cuando se lee, y por lo general no conservan los valores numéricos exactos de los coeficientes.

[1]Después de que un modelo MIP ha sido resuelto, se puede calcular el modelo fijo asociado. Este modelo es idéntico al modelo de entrada, excepto que todas las variables enteras se fijan a sus valores en la solución MIP.

Por otro lado, los saltos de línea y espacios en blanco se utilizan para objetos separados. A continuación, se muestra un ejemplo sencillo [46]:
\ Ejemplo de archivo LP

Maximize
$x + y + z$
Subject To
$c0 : x + y = 1$
$c1 : x + 5y + 2z <= 10$
Bounds
$0 <= x <= 5$
$z >= 2$
Generals
xyz
End

El símbolo de diagonal invertida se utiliza para iniciar un comentario. Los nombres de variables juegan un papel importante en los archivos de LP. Cada variable debe tener su propio nombre. El nombre no debe tener más de 255 caracteres, y para evitar confundir al analizador LP, no debe comenzar con un número o con cualquiera de los símbolos $+, -, *, <, >, =, o$:.

Se debe de tener en cuenta que el espacio en blanco no es opcional en el formato LP, por ejemplo, el texto $x+y+z$ sería tratado como un único nombre de la variable, mientras que $x + y + z$ es como una expresión de tres términos. Los archivos LP se estructuran como una lista de secciones, donde cada sección captura una pieza de todo el modelo de optimización. Las secciones comienzan con palabras clave en particular, y en general deben llegar en un orden fijo, aunque a algunas se les permite ser intercambiadas.

2.2.2.1.-Sección objetivo. La primera sección en un archivo de LP es la sección objetivo. Esta sección comienza con una de las siguientes seis cabeceras *minimize , maximize , minimum , maximum , min , o max*. El objetivo comienza opcionalmente con una etiqueta. Una etiqueta consta de un nombre, seguido de un caracter de dos puntos, seguido de un espacio. El objetivo continúa con una lista de términos lineales, separados por el $+$ o $-$. Un término puede contener un coeficiente y una variable (por ejemplo, 4.5x), o simplemente una variable (por ejemplo, x). El objetivo se puede distribuir en muchas líneas, o puede estar incluido en una sola línea.

El objetivo puede estar conformado por una lista de términos cuadráticos (si es el caso). La parte cuadrática de la expresión objetivo comienza con un símbolo [y termina con un], seguido de $/2$. Estos deben incluir uno o más términos de segundo grado. Ya sea en términos al cuadrado (por ejemplo, 2 x $\wedge$ 2), o términos de productos (3 x * y).

2.2.2.2.-Sección de restricciones. Se inicia con una de las siguientes cabeceras: *subject to, such that, st, o st*. La sección de restricciones puede tener un número arbitrario de restricciones. Cada restricción comienza con una etiqueta opcional (nombre de restricción, seguido de dos puntos, seguido de un espacio), continúa con una expresión lineal, seguido de una expresión cuadrática opcional (entre corchetes), y termina con un operador de comparación, seguido de un valor numérico, seguido por un salto de línea. Los operadores de comparación válidos son: $=, <=, <, >=, o >$. Se debe tener en cuenta que el lado izquierdo de una restricción no puede contener un término constante; la constante debe aparecer siempre en el lado derecho.

2.2.2.3.-Sección límites. Comienza con la palabra *Bounds*, y es seguido por una lista de límites variables. Cada línea especifica el límite inferior, el límite superior, o ambos para una sola variable. Las palabras clave inf o $infinity$ se pueden utilizar en la sección de grada para especificar límites infinitos, $free$ significa que la variable no está acotada en cualquier dirección. No es necesario especificar límites para todas las variables; de forma predeterminada, a cada variable se le asigna un valor mayor o igual que 0 y menor que infinito.

2.2.2.4.-Sección tipo de variable. Las variables pueden ser declarada como *binary, general integer, o semi-continuous*. En todos los casos, la designación se aplica proporcionando en primer lugar en el encabezado correspondiente y, seguido de esto, una lista de las variables que tienen el tipo asociado.

Capítulo 4

Estrategias Desarrolladas

1.- Adaptaciones de las técnicas base para resolver el *VRP-TW*

A continuación, se describen todas las adaptaciones que se realizaron en este trabajo, para resolver el *VRP-TW*.

1.1.-Sistema de hormigas

Para este problema, en cada iteración *a* se lanza un conjunto de *n* hormigas, las cuales construyen una solución guiada por la probabilidad de elección de visitar al cliente *j* después de visitar al cliente *i*.

Antes de ejecutar el primer ciclo para las *n* hormigas, se utiliza la función de inicialización de feromona y actualización de la probabilidad de elección. Una vez terminada la construcción de las *n* hormigas, se elige la mejor solución y se actualiza el nivel de feromona para cada camino que une al cliente *i* con el cliente *j* que se encuentre en esta.

Para que una hormiga construya una solución se le asocia una estructura de datos que contiene la información acerca de las ventanas de tiempo para cada cliente, una matriz de distancias d_{ij} (distancia entre el cliente *i* y el cliente *j*), una matriz de feromonas (que denota el nivel de feromona que hay entre un camino que une al cliente *i* con el cliente *j*), y una matriz de probabilidad de elección del camino que une al cliente *i* con el cliente *j*.

Además, en este trabajo se realizó la adaptación de la fórmula del cálculo de la probabilidad de elección (ecuación (4.4)), tomando en cuenta la particularidad del problema respecto a los límites de la ventana de tiempo para cada cliente. A continuación, se listan las principales funciones de la metaheurística *AS*.

1.1.1.- Inicialización de Feromonas (en términos de *VRP-TW*)

Para el primer ciclo se inicializa la matriz de nivel de feromona como se muestra en la ecuación (4.1):

$$\tau_{ij}^{a+1} = 1/l_j, \text{ con } a = 0 \tag{4.1}$$

Donde, τ_{ij}^{1} es el nivel de feromona que tiene el camino entre el cliente *i* y el cliente *j* para el ciclo *1*, y l_j el último momento en que se puede atender al cliente *j*.

1.1.2.- Actualización de Feromonas (en términos de *VRP-TW*)

Para los demás ciclos la matriz de nivel de feromona se actualiza de la forma que se ilustra en la ecuación (4.2):

$$\tau_{ij}^{a+1} = \rho(\tau_{ij}^{a}) + \Delta_{ij} \tag{4.2}$$

Donde, ρ es el coeficiente de evaporación de feromona, τ_{ij}^{a} es el nivel de feromona que tiene el camino entre el cliente *i* y el cliente *j* para el ciclo *a*, τ_{ij}^{a+1} es el nivel de la feromona que guiará al siguiente ciclo de las *n* hormigas y Δ_{ij} (ecuación (4.3)) como la cantidad de feromonas depositadas por la mejor hormiga del ciclo:

$$\Delta_{ij} = \frac{sol[a] - sol[a-1]}{sol[a-1]} \tag{4.3}$$

Si el camino de *i* a *j* está en la mejor hormiga del ciclo *a*. Siendo sol[*a*], el valor de la solución entregada por la mejor hormiga del ciclo *a*; y sol[*a*-1], el valor de la solución entregada por la mejor hormiga del ciclo *a-1* [28].

1.1.3.- Cálculo de la Probabilidad (en términos de *VRP-TW*)

Cada vez que se termina un ciclo *a* de *n* hormigas o, se agrega un cliente a la solución, se actualiza la probabilidad de elegir el camino de un cliente *i* a un cliente *j* mediante la siguiente ecuación:

$$P_{ij}^{a+1} = \frac{(\tau_{ij}^{a+1})^{\alpha}(\eta_{ij})^{\beta}(\nu_{ij})^{\gamma}}{\sum(\tau_{ij}^{a+1})^{\alpha}(\eta_{ij})^{\beta}(\nu_{ij})^{\gamma}} \tag{4.4}$$

Donde, τ_{ij} es el nivel de feromona, $\eta_{ij} = 1/d_{ij}$ regula la conveniencia de viajar entre los clientes *i* y *j*, $\nu_{ij} = 1/l_j$ denota que entre menor sea el valor del último momento en que se puede visitar, la probabilidad de visitarlo al inicio será mayor. Además α se utiliza para controlar la influencia del nivel de feromona, β es un parámetro para controlar la influencia de la cercanía entre

clientes y γ controla la influencia del valor de l_j. Finalmente, se normalizan estas probabilidades de ir desde el cliente *i* hacia los demás clientes elegibles en ese momento, y se forma una ruleta ordenando de mayor a menor estos valores como se ve en la ecuación (4.5).

$$Pn_{ij}^{a+1} = Pn_{i,j-1}^{a+1} + P_{ij}^{a+1} \tag{4.5}$$

1.1.4.- Construcción de la Solución

Cada hormiga del sistema construye una solución partiendo del depósito, agregando a la ruta un cliente a la vez con base en la probabilidad de elección de cada uno (ecuación (4.4)) partiendo del cliente que fue agregado anteriormente a esta (Pseudocódigo 3).

Pseudocódigo 3 Construcción de soluciones

1: **mientras** no se satisfaga la capacidad del vehículo o no se pueda agregar un elemento más **hacer**
2: *Inicio*
3: Generar un número "Aleatorio" continuo entre 0 y 1
4: **si** "Aleatorio" $\leq Pn_{i,j}$ y "Aleatorio" $> Pn_{i,j-1}$ **entonces**
5: Agregar el cliente *j* a la ruta del vehículo *k*.
6: Modificar capacidad del vehículo *k* y calcular tiempo de finalización de servicio para *j*.
7: Actualizar la probabilidad de elección
8: **fin si**
9: **fin mientras**
10: Agregar el nodo depósito al final de la ruta *k* (es necesario agregarlo la cantidad restante de elementos disponibles en el vector)

Finalmente, se tendrá una solución que se representa mediante el siguiente vector:

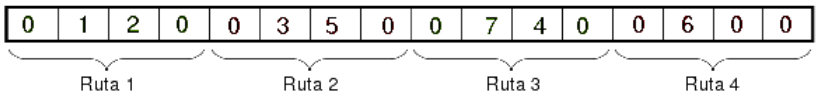

FIGURA 4.1: Vector solución.

De esta forma todas las rutas tendrán el mismo número de posiciones en el vector solución. Es decir, cada ruta tendrá asignado en el vector solución, un número *e* de espacios, el cual está dado por el número de clientes de la ruta con mayor número asignado de estos, más los espacios que representan el nodo depósito en salida y llegada.

A continuación, se muestra el Pseudocódigo 4 para la técnica *AS*.

Pseudocódigo 4 Sistema de hormigas

```
1: Inicializar Feromonas utilizando la ecuación (4.1)
2: Actualizar la probabilidad de elección para el primer ciclo de n hormigas
3: para a=1 hasta a=m hacer
4:     para l=1 hasta l=n hacer
5:         Construir Solución de la hormiga l
6:         Evaluar el Valor de Función Objetivo
7:     fin para
8:     Buscar la mejor solución entre las n soluciones generadas
9:     Guardar en memoria la mejor solución del ciclo a
10:    Actualizar Feromona con base en la mejor solución encontrada en el
       ciclo a utilizando la ecuación (4.2)
11:    Actualizar Probabilidad de elección utizando la ecuación (4.4)
12: fin para
13: Buscar la mejor solución entre las mejores de cada ciclo
14: devolver La mejor solución encontrada
```

En la Figura 4.2 se ilustra la idea principal de la técnica Sistema de Hormigas, donde se puede apreciar la forma en que las hormigas siguen los caminos, concentrándose en su mayoría en los caminos más cortos, dados por la mayor acumulación de la feromona:

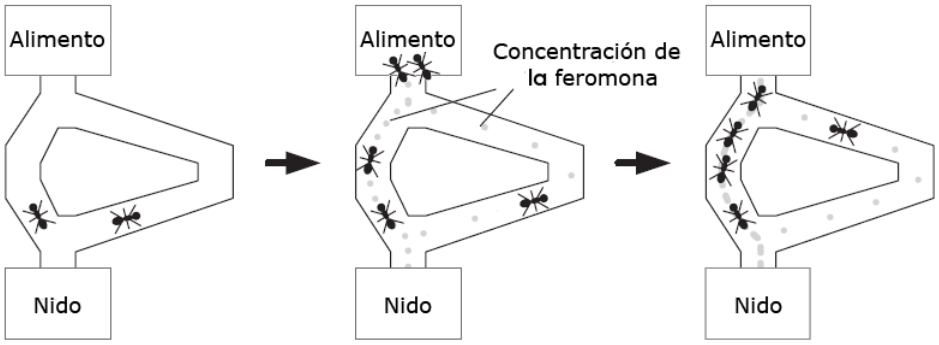

FIGURA 4.2: Procedimiento técnica Sistema de Hormigas.

1.2.- Búsqueda Local Iterada

En *ILS*, se logra salir de mínimos locales al aplicar perturbaciones al mínimo local una vez que se estanca en este. La fuerza de la perturbación está dada por el número de elementos de la solución que son cambiados. Para permitir escapar de mínimos locales, la perturbación debe ser lo suficientemente fuerte para ayudar a la búsqueda a encontrar el nuevo y posible mejor mínimo local, pero no tan fuerte para evitar una gran diversificación. Además, la perturbación aplicada debe ayudar al algoritmo de búsqueda a alcanzar el próximo óptimo local en pocos pasos.

1.2.1.- Perturbaciones

En este trabajo, para lograr un equilibrio entre la diversificación e intensificación dentro de ls búsqueda local se utilizaron 3 perturbaciones de diferente fuerza, las cuales se definen a continuación:

1. **Fuerte:** Perturbación *movimiento doble-puente*. Esta perturbación se considera fuerte debido a que puede ocasionar la aleatoriedad de las soluciones y diversificación del espacio de búsqueda. Se toman cuatro clientes en una ruta, los cuales se intercambian entre ellos formando a lo más dos puentes en la solución (debido a que se considera la restricción de ventana temporal de cada cliente).

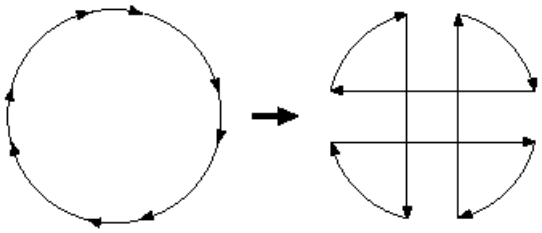

FIGURA 4.3: Perturbación movimiento doble-puente.

2. **Media:** Perturbación *cadena de expulsión*. Esta perturbación se considera como *de las mejores perturbaciones* para el tipo de problemas *VRP*. En esta perturbación, se intercambian clientes entre distintas rutas, para lo cual se analiza primero si al realizar el intercambio se siguen cumpliendo las restricciones de temporalidad y capacidad, si no se cumplen se busca una ruta en la cual se cumplan estas.

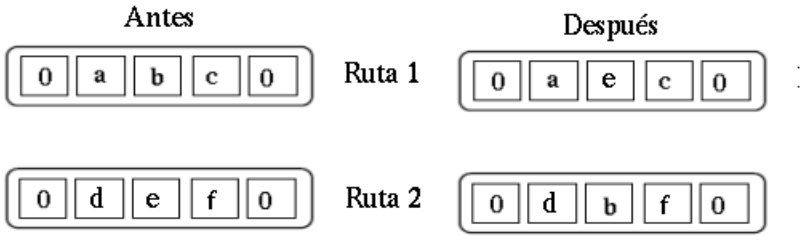

FIGURA 4.4: Perturbación cadena de expulsión.

3. **Débil:** Perturbación *doble intercambio*. Esta perturbación se considera débil, ya que realiza el intercambio de dos clientes dentro de la ruta (de forma aleatoria), y dependiendo del vecindario utilizado en la *LS* puede ser que se visite un óptimo local visitado con anterioridad.

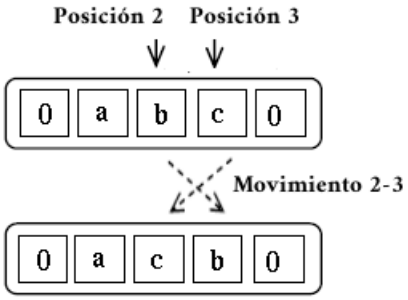

FIGURA 4.5: Perturbación doble intercambio.

Para la elección de la perturbación que se aplicará a la solución que se encuentra atorada en un óptimo local, se asigna mayor probabilidad a la perturbación media después de cierto número de iteraciones, mientras que se le da mayor probabilidad de elección en las primeras iteraciones a la perturbación fuerte, ya que con esto se logra diversificar al inicio de la búsqueda, y una vez que se llega a las últimas iteraciones se le da mayor probabilidad de elección a la perturbación débil, para poder así intensificar la búsqueda.

1.3.- Búsqueda armónica

Para este trabajo, la memoria armónica *HM* utilizada en *HS* se trabaja como la lista de *n* soluciones (armonías) generadas aleatoriamente, la cual se va modificando dependiendo del número de improvisaciones y el número de iteraciones dadas. El ancho de desplazamiento *(BW)* se adapta conforme el paso de las iteraciones, esto es, durante el transcurso de la ejecución toma valores que permiten perturbar de dos diferentes formas las soluciones, ya sea aplicando perturbación suave y/o media, por lo que deja de ser un parámetro a calibrar. Los demás parámetros como la razón de exploración (*HMCR*), la razón de ajuste de tono *(PAR)*, el ancho de desplazamiento *(BW)* y el número de improvisaciones *(NI)* deben calibrarse para un correcto funcionamiento de la técnica.

1.3.1.- Inicialización de la memoria armónica

Para este procedimiento dentro de la técnica, la *HM* se llena de manera aleatoria con *HMS* soluciones. Esta memoria se almacena en forma de lista, la cual contiene las x_i soluciones, con $i = 1, ..., HMS$ donde cada una cuenta con los elementos de la solución y el valor de la función objetivo. En la Figura 4.6 se muestra la caracterización de *HM*.

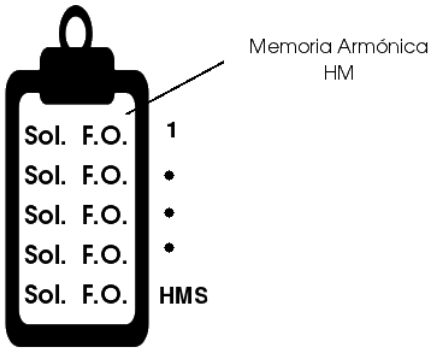

FIGURA 4.6: Memoria armónica.

1.3.2.- Improvisación de nuevas armonías

En esta etapa, una nueva solución x_s es improvisada, con base en los siguientes operadores:

1. Examinado de memoria.
2. Re-inicialización aleatoria.
3. Ajuste de tono.

Como se muestra en la Figura 4.1, cada solución se divide en *k* rutas. Entonces, para el paso de examinado de memoria, se genera un número aleatorio r_1 con una distribución uniforme en [0,1]. Si r_1 es menor que *HMCR* la ruta *i* para la nueva solución (improvisación) es elegida de cualquiera de las soluciones existentes dentro de *HM* (del conjunto $\{x_1(i), x_2(1), ..., x_{HMS}(1)\}$). De otro modo, $x_s(i)$ se obtiene a partir de una re-inicialización, por lo cual, para este trabajo se construye la ruta utilizando los índices para la feromona inicial de *AS*. Este proceso se realiza para cada una de las rutas.

Una vez que se lleva a cabo la re-inicialización aleatoria o el examinado de memoria, gracias a la razón de ajuste de tono se modifica la solución x_s mediante el valor de *(BW)*, realizando cambios en la solución utilizando los vecindarios descritos en las perturbaciones de la Subsección 4. A continuación, se muestran los dos tipos de perturbaciones:

En el Pseudocódigo 5 se muestra el procedimiento computacional para la etapa de improvisación y en el Pseudocódigo 6 se muestra la forma de elección de perturbación dependiendo del valor de *BW*. Se hace notar que el valor *BW* se adaptará conforme el paso de las iteraciones, esto es, durante el transcurso de la ejecucion irá tomando valores que permitirán perturbar en los dos vecindarios mostrados anteriormente.

Pseudocódigo 5 Improvisación de armonía

```
si r_1 ≤ HMCR entonces
    x_s = x_{r_2}
si no
    x_s = Genera solución aleatoria
fin si
si r_3 ≤ PAR entonces
    Perturbar x_s
fin si
```

Donde $r_2 \in (1, HMS)$, $r_1, r_3 \in (0, 1)$, *HMCR* es la razón de exploración, *PAR* .

Pseudocódigo 6 Elección de perturbaciones

```
para j=1 hasta j = n_rutas hacer
    si r_3 ≤ PAR entonces
        si BW == 1 entonces
            Perturbar x_s con doble intercambio
        si no
            Perturbar x_s con cadena de expulsión
        fin si
    fin si
fin para
```

Terminada la generación (improvisación) de la nueva armonía, se realiza la evaluación de la función objetivo, para después poder compararla respecto a la información guardada en *HM*.

1.3.3.- Actualización de la memoria armónica

Después de generar una nueva armonía x_s, la memoria *HM* es actualizada comparando el valor de la función objetivo entre x_s y la peor solución x_w contenida en la memoria *HM*. Entonces, si el valor de la función objetivo de x_s es mejor que x_w, x_s lo remplazará, en caso contrario el contenido de *HM* permanece sin cambios.

Gracias a que *HM* contiene a los elementos cuyo valor de la función objetivo representa a las mejores improvisaciones, al paso de las iteraciones se almacenará sólo a la mejor solución o a soluciones que se encuentren dentro del vecindario de la misma.

A continuación, se muestra el Pseudocódigo 7 que denota la forma de actualización de *HM*.

Pseudocódigo 7 Actualización de HM

si $x_s \leq x_w$ **entonces**
    $x_w = x_s$
si no
    No se realizan cambios
fin si

En el Pseudocódigo 8 se muestra el procedimiento principal de la técnica, y para ilustrar la forma en que se ejecuta, la Figura 4.7 detalla cómo se actualiza la memoria armónica con base en las improvisaciones generadas.

Pseudocódigo 8 Búsqueda Armónica

Generar *HMS* soluciones aleatorias
mientras $n\_imp \leq NI$ **hacer**
    si $n\_imp \leq 1/3 \times NI$ **entonces**
        BW=1
    si no si $1/3 \times NI \leq n\_imp \leq 2/3 \times NI$ **entonces**
        BW=2
    si no
        BW=1
    fin si
    Improvisar una nueva armonía x_s utilizando el Pseudocódigo 5
    Llamar al Pseudocódigo 7
fin mientras
Ordenar ascendentemente las *HMS* soluciones por valor de F.O.
devolver La mejor solución encontrada

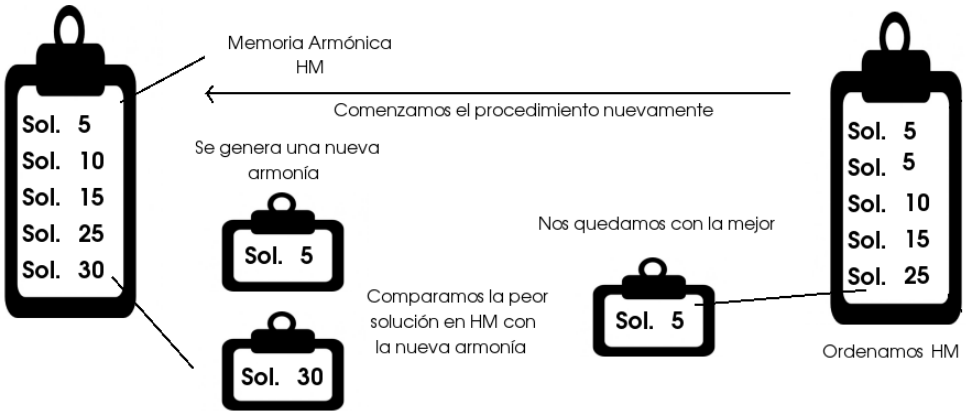

FIGURA 4.7: Procedimiento principal búsqueda armónica.

1.4.- Algoritmo genético

En este trabajo, a diferencia de [51] en *GA* se generan n soluciones aleatorias a partir de los índices $1/lj$, las cuales se trabajan como la primera generación, en donde, sus individuos se modifican en cada generación mediante los operadores genéticos. Una vez que los operadores genéticos se aplican a la población, para evitar generar soluciones infactibles del problema, cada uno de los individuos pasa por un proceso de reparación. Estos procesos se llevan a cabo de manera iterativa, hasta que se cumpla el número de generaciones dadas para el algoritmo.

1.4.1.- Selección

El proceso recibe una población de individuos, de la cual se selecciona un número predeterminado de individuos con base en el criterio de "elitismo", los cuales pasarán directamente a la siguiente generación, y los individuos restantes se generarán a partir de los operadores de cruza y mutación. Este proceso se describe en el Pseudocódigo 9 [16]:

Pseudocódigo 9 Selección en GA

```
1: para l=1 hasta l=n hacer
2:     Evaluar la función objetivo para la solución l.
3:     Ordenar las soluciones con base en el valor de la función objetivo.
4: fin para
5: para l=1 hasta l=n*Elitismo hacer
6:     Copiar el elemento l de la población a la nueva generación en la posición
   l.
7: fin para
8: para l=n*Elitismo hasta l=n hacer
9:     Generar un número aleatorio entre 0 y n.
10:    Copiar el elemento Aleatorio de la población a la nueva generación en la
   posición l.
11: fin para
```

Es importante destacar que se debe ajustar el parámetro de *Elitismo*, ya que el desempeño de la técnica depende en parte de este valor.

1.4.2.- Cruza

Este proceso, recibe los individuos que fueron seleccionados en el proceso anterior. Aquí se eligen de forma aleatoria dos individuos, llamados *padres*, que se recombinan formando dos nuevos individuos, llamados *hijos*, los cuales contienen información genética de ambos padres; esto se lleva a cabo hasta

completar el número de nuevos individuos necesarios para completar una población de tamaño n.

Por el proceso de selección "Elitista" se pasa directamente un porcentaje de la población a la nueva generación, y el porcentaje restante se genera combinando información de dos *padres* seleccionados aleatoriamente. Debido al tipo de problema, la cruza se lleva a cabo en k puntos (donde k es el número de vehículos), para el primer hijo se toman las rutas *impares* correspondientes al padre 1 y a las rutas *pares* correpondientes al padre 2; y para el segundo hijo se toman las rutas *pares* correpondientes al padre 1 y las rutas *impares* correspondientes al padre 2. Entonces al final de este procedimiento se tendrán los dos hijos formados con información de ambos padres. En las Figuras 4.8 y 4.9 se muestra como se forman los hijos a partir de los padres de la siguiente forma [16]:

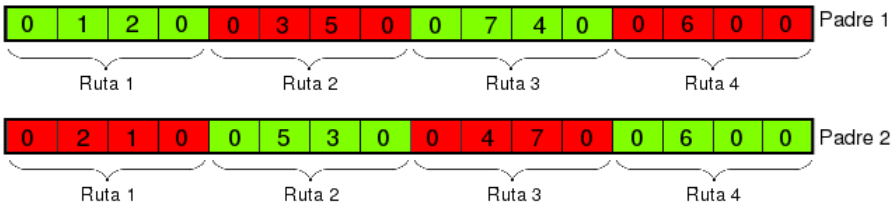

FIGURA 4.8: Elementos a cruzar.

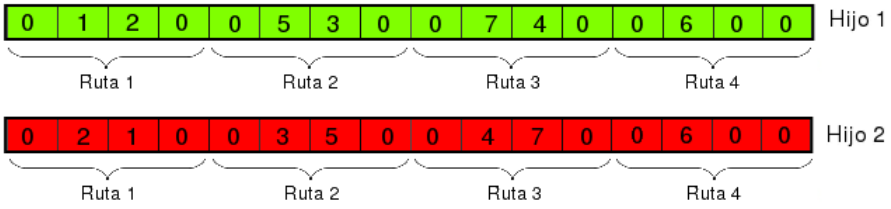

FIGURA 4.9: Elementos generados por la cruza.

Para el ejemplo mostrado se tiene que los clientes asignados a las rutas en ambos padres son correspondientes, sin embargo no siempre sucede; por lo que se debe llevar a cabo un proceso de reparación (se marca con color gris las rutas que serán asignadas al hijo 1, y con color negro las rutas que serán asignadas al hijo 2).

1.4.3.- Mutación

Este proceso recibe la población de nuevos individuos (compuesta por los individuos de élite seleccionados y los individuos *hijo* formados en el proceso

de cruza); entonces se "perturba" cada uno de estos nuevos individuos siguiendo el Pseudocódigo 10:

Pseudocódigo 10 Mutación en GA

```
1: para l=1 hasta l=clientes hacer
2:     Generar un número "Aleatorio" entre 0 y 1
3:     si "Aleatorio" ≤ tasa_mutación entonces
4:         Intercambiar ese cliente con otro elegido de forma aleatoria.
5:     fin si
6: fin para
```

A continuación se ilustra el movimiento de mutación:

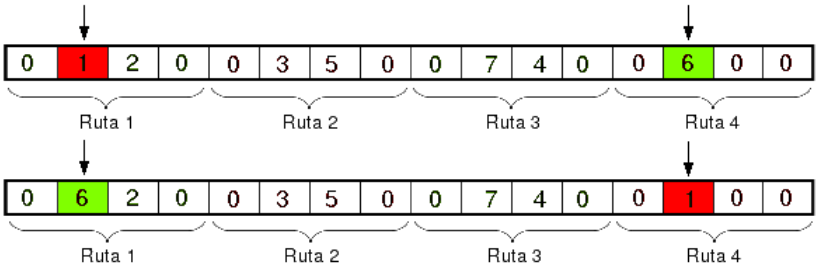

FIGURA 4.10: Movimiento de mutación.

1.4.4.- Reparación

Para evitar la infactibilidad en los individuos que formarán parte de la siguiente generación, se revisa que cumplan con las restricciones del problema. Este proceso se describe en el Pseudocódigo 11:

Pseudocódigo 11 Reparación en GA

```
1: para l=1 hasta l=n hacer
2:     Recorrer los elementos (clientes) de la solución l
3:     si clientes_repetidos==VERDADERO entonces
4:         Eliminar los clientes repetidos de la solución l
5:         Revisar si hay elementos faltantes en la solución l
6:         si Elementos_faltantes==VERDADERO entonces
7:             Agregar un faltante en cada "espacio" dejado por los repetidos
   sin violar la capacidad del vehículo en cuestión
8:         fin si
9:     si no
10:        Revisar si aún tenemos elementos faltantesl
11:        si Elementos_faltantes==VERDADERO entonces
12:            Agregar un elemento faltante en la posición l
13:        fin si
14:    fin si
15: fin para
```

El procedimiento general del algoritmo genético se puede ver en el Pseudocódigo 12:

Pseudocódigo 12 Algoritmo Genético

```
1: Recibir población inicial
2: para a=1 hasta a=generaciones hacer
3:     Selección
4:     Cruza
5:     Mutación
6:     Reparación
7:     Evaluación
8: fin para
```

2.- Algoritmos híbridos para el *VRP-TW*

Un algoritmo híbrido, es una técnica donde se combinan dos o más algoritmos (sean exactos o heurísticos) los cuales se utilizan para solucionar un cierto problema. En estas técnicas, se busca combinar las mejores características de cada uno, con el fin de que la última técnica mejore a las técnicas utilizadas en cuestión. En el presente capítulo, se describen los algoritmos híbridos realizados en este trabajo para resolver el *VRP-TW*.

Al utilizar las técnicas descritas en el Capítulo 3 con las modificaciones denotadas anteriormente, se realizaron cuatro algoritmos híbridos, tomando como base la técnica *AS*, combinando con esta las mejores características de *ILS*,

GA, *HS*, *DSM* y *PDA*. En esta sección se describe la forma en que se realizaron estas hibridaciones.

2.1.- Algoritmo híbrido sistema de hormigas-búsqueda local iterada (AS-ILS)

Durante las pruebas de las técnicas descritas en la sección anterior, se observó que la metaheurística *AS* en algunas instancias regresaba como mejor solución, una que se encontraba dentro de algún vecindario cercano a la mejor solución reportada en la literatura. Entonces, se decidió implementar una búsqueda local iterada después de los *a* ciclos de *n* hormigas, logrando alcanzar en mayor número de ocasiones la mejor solución reportada. A continuación, se muestra el Pseudocódigo de la técnica denominada *AS-ILS*.

Pseudocódigo 13 Sistema de hormigas-Búsqueda Local Iterada

1: Inicializar feromonas utilizando la ecuación 4.1
2: Actualizar la probabilidad de elección para el primer ciclo de *n* hormigas
3: **para** *a=1* **hasta** *a=m* **hacer**
4: **para** *l=1* **hasta** *l=n* **hacer**
5: Construir solución de la hormiga *l*
6: Evaluar la función objetivo
7: **fin para**
8: Buscar la mejor solución entre las *n* soluciones generadas.
9: Guardar en memoria la mejor solución del ciclo *a*
10: Actualizar feromona con base en la mejor solución encontrada en el ciclo *a* utilizando la ecuación 4.2
11: Actualizar probabilidad de elección utizando la ecuación 4.4
12: **fin para**
13: $S =$ Mejor solución entre las mejores de cada ciclo
14: Aplicar un algoritmo de búsqueda, que proporcione un óptimo local S^* utilizando el Pseudocódigo 1.
15: **mientras** no se cumpla el criterio de parada **hacer**
16: Aplicar una perturbación a la solución S^* para transformarla en S'
17: Emplear el algoritmo de búsqueda para obtener $S^{*'}$
18: Si $S^{*'}$ supera un criterio de aceptación, considerar a $S^{*'}$ como el siguiente S^*
19: **fin mientras**
20: **devolver** la solución *S*

2.2.- Algoritmo híbrido sistema de hormigas-búsqueda armónica (AS-HS)

Para el desarrollo de esta técnica, se utilizaron dos metaheurísticas descritas en la sección anterior (*AS* y *HS*) trabajando de manera entrelazada [79].

Con lo cual se pretende guiar la construcción de las soluciones de cada hormiga, aprovechando las mejores melodías generadas por *HS*, ya que se tomarán las *m* soluciones de la lista *HM* que se encuentren por encima del valor promedio para actualizar la feromona para el *AS*. Con lo cual, se diversificará el espacio de búsqueda tomando soluciones que se encuentren dispersas. En el Pseudocódigo 14 se muestra el procedimiento general de esta hibridación.

Pseudocódigo 14 Sistema de Hormigas-Búsqueda Armónica

```
Inicializar Feromonas con una suma ponderada de (1/d y 1/TW)
Actualizar la probabilidad de elección para el primer ciclo de HMS hormigas
para a=1 hasta a=m hacer
    si a ≤ 1/3 × m entonces
        BW=1
    si no si 1/3 × m ≤ a ≤ 2/3 × m entonces
        BW=2
    si no
        BW=1
    fin si
    si a==1 entonces
        para l=1 hasta l=HMS hacer
            Construir Solución de la hormiga x_l
            Evaluar el Valor de Función Objetivo
            Guardar en la memoria HM la solución x_l
        fin para
        Ordenar ascendentemente las HMS soluciones por valor de F.O.
    si no
        para l=contador hasta l=HMS hacer
            Construir Solución de la hormiga x_l
            Evaluar el Valor de Función Objetivo
            Guardar en la memoria HM la solución x_l
        fin para
        contador==0
    fin si
    mientras n_imp ≱ NI hacer
        Improvisar una nueva armonía x_n
        si x_HMS > x_n entonces
            x_HMS = x_n
        si no
            x_HMS = x_HMS
        fin si
    fin mientras
    Ordenar ascendentemente las HMS soluciones por valor de F.O.
    Calcular promedio de las HMS soluciones
    para l=1 hasta l=HMS hacer
        si x_l ≤ promedio entonces
            Actualizar Feromona con base en x_l
            Actualizar Probabilidad de elección
            Aumenta contador
        fin si
    fin para
    Guardar la solución x_1 en memoria de tamaño a
fin para
Buscar la mejor solución entre las mejores de cada ciclo
devolver La mejor solución encontrada
```

Donde:
a, es el número de ciclos para el *AS*.
HMS, es el tamaño de la memoria armónica (HM) y representa el número de hormigas.
1/d, es la inversa de la distancia entre dos clientes.
1/TW, es la inversa del límite final de la ventana de tiempo para cada cliente.

2.3.- Algoritmo híbrido sistema de hormigas-algoritmo genético (AS-GA)

Para el desarrollo de este algoritmo híbrido, se utilizaron las metaheurísticas *AS* y *GA* ya descritas; trabajando de manera entrelazada [79]. Para esto, en el procedimiento general de *AS* una vez terminado un ciclo *a*, se mandan las soluciones dadas por las *n* hormigas como población del *GA*, el cual lleva a cabo sus operadores genéticos con esta población y devuelve la mejor solución obtenida después de una generación; entonces, con esta solución devuelta por el algoritmo genético se lleva a cabo el proceso de *Actualización de Feromona* para el *AS*, en el Pseudocódigo 15 se muestra la realización de esta adaptación:

Pseudocódigo 15 Sistema de Hormigas-Algoritmo Genético (AS-GA)

```
1: Inicializar Feromonas utilizando la ecuación 4.1
2: Actualizar la probabilidad de elección para el primer ciclo de n hormigas
   utilizando las ecuaciones y 4.4
3: para a=1 hasta a=m hacer
4:     para l=1 hasta l=n hacer
5:         Construir Solución de la hormiga l
6:         Evaluar la función objetivo
7:         Guardar en memoria cada solución l
8:     fin para
9:     Crear lista con las n soluciones que serán la población inicial para el GA
10:    Llamar a Algoritmo Genético
11:    Actualizar Feromona con base en la mejor solución encontrada por el
   GA utilizando la ecuación 4.2
12:    Actualizar Probabilidad de elección utilizando la ecuación 4.4
13: fin para
14: Buscar la mejor solución entre las mejores de cada ciclo
15: devolver La mejor solución encontrada
```

De esta forma, se busca que el *GA* combine las mejores características de las soluciones obtenidas por *AS*, dando como resultado una mejor solución, misma que retroalimentará a *AS* mediante la actualización de feromonas.

2.4.- Algoritmo híbrido sistema de hormigas, algoritmo primal-dual y método dual simplex (DSM-AS-PDA)

Esta técnica al ser una hibridación entre métodos de programación matemática y técnicas heurísticas, se encuentra dentro de la clase de las *matheurísticas, (matheuristics)*; donde, una característica esencial de las matheurísticas es la explotación en alguna parte de los algoritmos de características derivadas del modelo matemático de los problemas de interés, por lo que la definición de "heurística basada en modelos" también es utilizada para estos algoritmos [65].

Para el desarrollo de esta matheurística se utilizaron las técnicas de método dual simplex, sistema de hormigas y algoritmo primal-dual. Para el método dual simplex se utilizó la herramienta *Gurobi* (la cual se describe en la siguiente sección) para resolver los programas lineales relajados apartir del programa entero para el *VRP-TW*. Una vez que *Gurobi* devuelve una solución se toma esa información y se inicializa el nivel de la matriz de *feromona* dentro de *AS*, con lo que se comienza la ejecución normal de la metaheurística con la instancia del *VRP-TW* original y después de un número m de iteraciones se ejecuta el método primal-dual para revisar la optimalidad de la solución encontrada. En caso de que no sea la solución óptima se llama nuevamente a *AS* un número b de iteraciones.

En el Pseudocódigo 16 se muestra el procedimiento general de esta hibridación.

Pseudocódigo 16 Método dual simplex-Sistema de hormigas-Algoritmo primal-dual

Entrada: Programa lineal original

1: Relajar el programa lineal original en términos de integralidad y restricciones temporales
2: Relajar el programa lineal original en términos de integralidad y restricción de capacidad vehicular
3: Resolver programas lineales relajados con *Gurobi*
4: Llamar a AS (Pseudocódigo 4)
5: Formar el programa lineal relajado en integralidad y restricciones temporales utilizando la solución devuelta por *AS*
6: Formar el programa dual del programa lineal relajado
7: A partir de una solución y factible para el programa dual, encontrar una x factible para el programa primal
8: **si** $x = y$ **entonces devolver** x como la mejor solución
9: **si no**
10: Llamar a AS (Pseudocódigo 4) y regresar a la línea 5
11: **fin si**

Salida: Solución del problema original

Para poder resolver el *VRP-TW* con la herramienta *Gurobi* en un tiempo no muy largo, es necesario relajar el modelo mostrado en la Ecuación 2.1. Para esto, en la siguiente subsección se muestra la manera en que se llevaron a cabo dos relajaciones, una respecto a integralidad y restricciones temporales y otra respecto a integralidad y capacidades vehiculares.

2.4.1.- Programas lineales relajados

2.4.1.1.-Relajación respecto a ventanas de tiempo e integralidad Se tomó como base el modelo mostrado en el Capítulo 2, y se realizaron dos tipos de relajaciones, con el fin de poder resolverlos utilizando la herramienta *Gurobi* en un tiempo corto. Entonces, la primer relajación de (2.1) respecto a integralidad y ventanas de tiempo se denota de la siguiente forma:

$$Minimizar \sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{v=1}^{k} C_{ij}X_{ij}^{v} \tag{4.6}$$

Sujeto a:

$$\sum_{i=0}^{n}\sum_{v=0}^{k} X_{ij}^{v} = 1;\ \forall\, j = 1, ..., n$$

$$\sum_{j=1}^{n} X_{0j}^{v} = 1;\ \forall\, v = 1, ..., k$$

$$\sum_{i=1}^{n} X_{i0}^{v} = 1;\ \forall\, v = 1, ..., k$$

$$\sum_{i=0}^{n} X_{ip}^{v} - \sum_{j=0}^{n} X_{pj}^{v} = 0;\ \forall\, p = 0, ..., n;\ v = 1, ..., k$$

$$Y_{iv} = di \sum_{j=0}^{n} X_{ij}^{v};\ \forall\, i = 1, ..., n;\ v = 1, ..., k$$

$$\sum_{i=0}^{n} Y_{iv} \leq Q\ \forall\, v = 1, ..., k$$

En (4.6), se han eliminado las restricciones que imponen las ventanas de tiempo, por lo que el problema se resolvería como el *VRP* tradicional, además al eliminar la restricción de integralidad se puede resolver como un programa lineal.

2.4.1.2.-Relajación respecto a capacidad vehicular e integralidad La segunda relajación de (2.1) respecto a integralidad y ventanas de tiempo se denota de la siguiente forma:

$$Minimizar \sum_{i=0}^{n}\sum_{j=0}^{n}\sum_{v=1}^{k} C_{ij}X_{ij}^{v} \tag{4.7}$$

Sujeto a:

$$\sum_{i=0}^{n}\sum_{v=0}^{k} X_{ij}^{v} = 1;\ \forall\, j = 1, ..., n$$

$$\sum_{j=1}^{n} X_{0j}^{v} = 1;\ \forall\, v = 1, ..., k$$

$$\sum_{i=1}^{n} X_{i0}^{v} = 1;\ \forall\, v = 1, ..., k$$

$$\sum_{i=0}^{n} X_{ip}^{v} - \sum_{j=0}^{n} X_{pj}^{v} = 0;\ \forall\, p = 0, ..., n;\ v = 1, ..., k$$

$$e_i \leq b_i^v \leq l_i\ \forall\, i, j = 0, ..., n;\ v = 1, ..., k$$

En (4.7), las restricciones que imponen las ventanas de tiempo han sido eliminadas, entonces el problema puede ser resuelto como un *Problema del reparador viajero con ventanas de tiempo* [1], además, al eliminar las restricciones de integralidad se puede resolver como un programa lineal.

[1]Este problema, consiste en un reparador que debe atender tareas o servicios, donde, cada tarea tiene asociada una ventana de tiempo indicando su disponibilidad de tiempo.

Capítulo 5

Resultados

1.- Metodología de pruebas

Para realizar el análisis sobre la robustez y el desempeño obtenido por las técnicas descritas en el Capítulo 3, se realizaron 20 ejecuciones para cada una de estas sobre 12 instancias de prueba para el *VRP-TW*.

1.1.- Instancias de Prueba

El conjunto de prueba se integró por 12 instancias de distintos tamaños; las cuales se tomaron del conjunto propuesto por Cordeau y Solomon [17, 18, 84] para el problema *VRP-TW*, las cuales han sido utilizadas por varios autores (e.g: [70, 89, 80]). En la Tabla 5.1 se muestran las instancias seleccionadas y el número de clientes involucrados.

Instancia	Clientes	Instancia	Clientes
C101	100	C105	50
C106	100	R201	50
C105	100	c106	50
C107	100	R202	50
C108	100	R201	25
C109	100	R202	25

TABLA 5.1: Casos de prueba.

Cada una de las instancias denota la cantidad de clientes, las coordenadas de cada cliente en x y y (con las cuales se calcula la distancia euclidiana), las veces que debe servirse, la demanda, los tiempos de inicio y final, el tiempo de servicio, el número de vehículos y la capacidad de estos. En la Tabla 5.2 se muestra el ejemplo de una instancia del tipo *VRP-TW* [17, 18, 84], tomando los primeros 10 clientes de la instancia $C101$.

Cliente	Coord. x	Coord. y	T. servicio	Demanda	Límite inicial de tiempo	Límite final de tiempo
0	40.00	50.00	0.00	0.00	1236	–
1	45.00	68.00	90.00	10.00	912	967
2	45.00	70.00	90.00	30.00	825	870
3	42.00	66.00	90.00	10.00	65	146
4	42.00	68.00	90.00	10.00	727	782
5	42.00	65.00	90.00	10.00	15	67
6	40.00	69.00	90.00	20.00	621	702
7	40.00	66.00	90.00	20.00	170	225
8	38.00	68.00	90.00	20.00	255	324
9	38.00	70.00	90.00	10.00	534	6605
10	35.00	66.00	90.00	10.00	357	410

TABLA 5.2: Instancia C101 (10 clientes).

1.2.- Ajuste de Parámetros

Como se describió en la Capítulo 3, las técnicas utilizadas necesitan de algunos parámetros de entrada que ayudan al desempeño de las mismas. El ajuste de parámetros es una actividad sumamente importante, con el objeto de lograr un comportamiento adecuado de las metaheurísticas.

En este trabajo se realizó un ajuste de los mismos con un algoritmo de *Evolución Diferencial (DE)*[1] [87]. Este algoritmo recibe como entrada los parámetros a ajustar como un vector. La longitud de cada uno de estos vectores es igual al número de parámetros más un espacio reservado para el valor de la aptitud, donde la población está compuesta de *NP* (Número de padres) vectores.

Además se necesita que el dominio de cada uno de los parámetros del problema esté restingido entre valores mínimo y máximo x_m^{min} y x_m^{max} con $m = 1, 2, ..., N$. El algoritmo de *DE* se compone de 4 pasos [87][76], los cuales se muestran a continuación:

1. Inicialización.
2. Mutación.
3. Recombinación.
4. Selección.

Donde, la incialización se lleva a cabo en la primer generación del algoritmo y los procesos de mutación-recombinación-selección se realizan para cada generación, hasta que se llegue al número máximo de generaciones. En la literatura, algunos autores recomiendan utilizar, los siguientes valores:

- Factor de cruza: 0.5.
- Factor de mutación: 0.5.

[1]Se realizó otra calibración mediante el algoritmo *F-Race*, sin embargo, *Evolución diferencial obtuvo mejores resultados.*

- Número de Generaciones: 10.

Para mostrar de mejor manera la forma en que se trabaja el ajuste de parámetros utilizando *DE* para las técnicas para resolver el *VRP-TW*, en la Figura 5.1 se describe el proceso general.

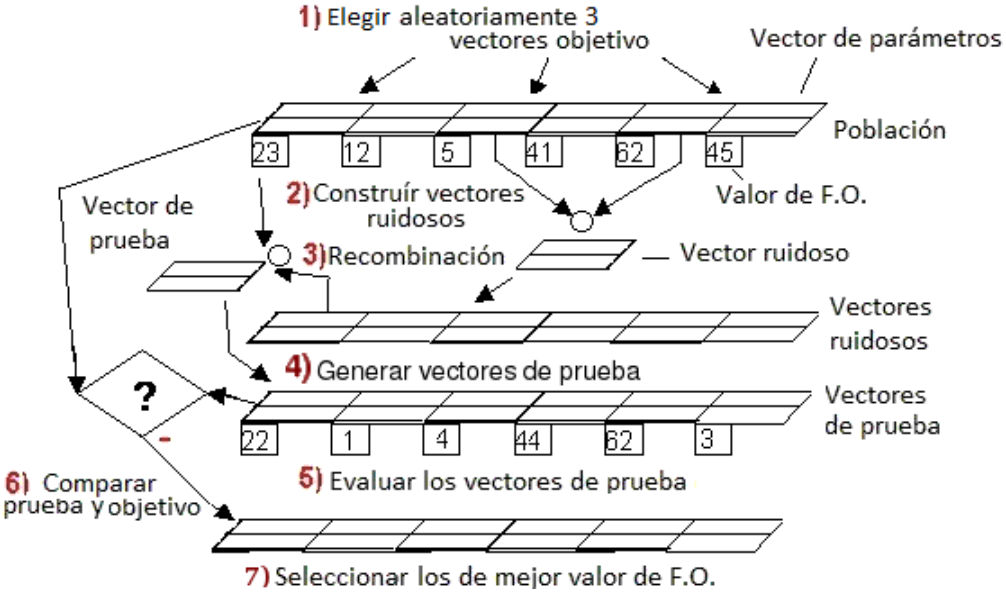

FIGURA 5.1: Ajuste de parámetros mediante *DE*.

La evaluación de la función objetivo para obtener la *aptitud* de los individuos en *DE* se realiza mediante una corrida de cada una de las técnicas aquí presentadas. Estas evaluaciones se realizaron 20 veces para cada una de las técnicas en instancias de tamaño grande. La configuración de parámetros obtenida para cada técnica se muestra en la Tabla 5.3.

Parámetro	*AS-GA*	*GA*	*AS*	*AS-ILS*	*HS*	*AS-HS*	*DSM-AS-PDA*
Número de iteraciones	25	–	200	200	100	50	50-60
Tamaño de la población	10	50	60	50	20	10	14
α	1	–	1	1	–	1.45	1.14
β	1.5	–	1.5	1.5	–	1	1.49
γ	1.3	–	1.3	1.3	–	1.55	1.35
ρ	0.75	–	0.75	0.75	–	0.75	0.71
Número de generaciones	10	150	–	–	–	–	–
Porcentaje de supervivencia	77 %	77 %	–	–	–	–	–
Tasa de mutación	10 %	10 %	–	–	–	–	–
Número de improvisaciones	–	–	–	–	5	6	–
Razón de exploración	–	–	–	–	0.673	0.6525	–
Razón de ajuste de tono	–	–	–	–	0.234	0.185	–
Llamadas a la Función Objetivo	2,500	7,500	12,000	10,000	10,000	3,000	1,000

TABLA 5.3: Configuración de parámetros.

En la Tabla 5.3 se puede observar que *DSM-AS-PDA* realiza menos llamadas a la función objetivo, sin embargo el tiempo de cómputo es mayor si es la primera vez que se corre *Gurobi*.

2.- Resultados

2.1.- Tablas de resultados

Para realizar un análisis de los resultados, cada técnica se ejecutó 20 veces sobre cada una de las 12 instancias de prueba [19, 84]; en donde en cada ejecución se registró el mejor valor de la función objetivo encontrado por la i-ésima técnica, así como el peor valor y promedio de los resultados obtenidos, además de la varianza y desviación estándar.

En las Tablas 5.4, 5.5, 5.6, 5.7, 5.8, 5.9 y 5.10 se muestra el mejor, peor y valor promedio obtenido por las metaheurísticas propuestos en las instancias consideradas y los mejores resultados reportados en [18, 70, 89, 80]. En estas se puede ver que todos las técnicas encontraron la mejor solución conocida, por lo tanto, para decidir cuál es la mejor opción se revisó la desviación estándar generada por cada uno; con este criterio de desempate se encontró que las opciones más robustas son los algoritmos híbridos *AS-GA* y *DSM-AS-PDA*. De forma adicional, *DSM-AS-PDA* requiere menos llamadas a la función objetivo, sin embargo al ser la primera vez que se ejecuta la herramienta *Gurobi* el tiempo de cómputo es más grande.

Instancia	Mejor reportado	Mejor	Peor	Promedio	σ^2	σ
C101	828.94	828.94	1171.31	888.99	15000.68	122.47
C105	828.94	828.94	846.73	835.03	54.49	7.38
C106	828.94	828.94	844.77	832.02	34.18	5.84
C107	828.94	828.94	844.77	834.38	46.96	6.85
C108	828.94	828.94	846.77	834.19	48.94	6.99
C109	828.94	828.94	844.77	830.35	8.11	2.84
C105	191.32	191.32	191.32	191.32	0	0
C106	191.32	191.32	191.32	191.32	0	0
R201	463.34	463.34	463.34	463.34	0	0
R202	410.55	410.55	410.55	410.55	0	0
R201	800.7	800.7	800.7	800.7	0	0
R202	712.25	712.25	712.25	712.25	0	0

TABLA 5.4: Resultados obtenidos por *AS*.

Instancia	Mejor reportado	Mejor	Peor	Promedio	σ^2	σ
C101	828.94	828.94	1120.83	915.25	15224.12	123.38
C105	828.94	828.94	841.41	835.65	17.24	4.15
C106	828.94	828.94	841.41	831.80	14.01	3.74
C107	828.94	828.94	841.41	835.63	33.73	5.80
C108	828.94	828.94	836.66	831.95	11.91	3.45
C109	828.94	828.94	830.60	830.05	0.64	0.80
C105	191.32	191.32	191.32	191.32	0	0
C106	191.32	191.32	191.32	191.32	0	0
R201	463.34	463.34	498.45	464.51	41.10	6.41
R202	410.55	410.55	410.55	410.55	0	0
R201	800.7	800.7	895.62	803.89	300.14	17.32
R202	712.25	712.25	712.25	712.25	0	0

TABLA 5.5: Resultados obtenidos por *HS*.

Instancia	Mejor reportado	Mejor	Peor	Promedio	σ^2	σ
C101	828.94	828.94	971.36	842.405	1691.89	41.13
C105	828.94	828.94	903.22	832.28	183.09	13.53
C106	828.94	828.94	920.62	837.34	661.43	25.71
C107	828.94	828.94	978.95	842.58	1771.5	42.09
C108	828.94	828.94	938.26	832.81	397.23	19.93
C109	828.94	836.66	832.14	829.36	1.23	1.109
C105	191.32	191.32	191.32	191.32	0	0
C106	191.32	191.32	191.32	191.32	0	0
R201	463.34	463.34	463.34	463.34	0	0
R202	410.55	410.55	410.55	410.55	0	0
R201	800.7	800.7	800.7	800.7	0	0
R202	712.25	712.25	712.25	712.25	0	0

TABLA 5.6: Resultados generados por *GA*.

Instancia	Mejor reportado	Mejor	Peor	Promedio	σ^2	σ
C101	828.94	828.94	1120.83	848.39	5484.36	74.05
C105	828.94	828.94	846.73	830.75	15.52	3.94
C106	828.94	828.94	830.6	829.88	0.70	0.84
C107	828.94	828.94	841.41	833.92	38.67	6.21
C108	828.94	828.94	830.60	830.05	0.64	0.80
C109	828.94	828.94	830.60	829.99	0.66	0.81
C105	191.32	191.32	191.32	191.32	0	0
C106	191.32	191.32	191.32	191.32	0	0
R201	463.34	463.34	463.34	463.34	0	0
R202	410.55	410.55	410.55	410.55	0	0
R201	800.7	800.7	800.7	800.7	0	0
R202	712.25	712.25	712.25	712.25	0	0

TABLA 5.7: Resultados obtenidos por *AS-ILS.*

Instancia	Mejor reportado	Mejor	Peor	Promedio	σ^2	σ
C101	828.94	828.94	1120.83	843.02	3322.70	57.64
C105	828.94	828.94	836.66	831.25	12.95	3.59
C106	828.94	828.94	836.66	830.13	7.03	2.65
C107	828.94	828.94	841.41	831.11	20.68	4.54
C108	828.94	828.94	836.66	830.13	7.03	2.65
C109	828.94	828.94	830.60	829.43	0.60	0.77
C105	191.32	191.32	191.32	191.32	0	0
C106	191.32	191.32	191.32	191.32	0	0
R202	410.55	410.55	410.55	410.55	0	0
R201	463.34	463.34	463.34	463.34	0	0
R201	800.7	800.7	800.7	800.7	0	0
R202	712.25	712.25	712.25	712.25	0	0

TABLA 5.8: Resultados generados por *AS-HS*.

Instancia	Mejor reportado	Mejor	Peor	Promedio	σ^2	σ
C101	828.94	828.94	828.94	828.94	0	0
C105	828.94	828.94	836.6	829.45	3.83	1.95
C106	828.94	828.94	830.6	829.27	0.46	0.67
C107	828.94	828.94	841.41	830.18	14.50	3.80
C108	828.94	828.94	830.6	829.04	0.18	0.42
C109	828.94	828.94	828.94	828.94	0	0
C105	191.32	191.32	191.32	191.32	0	0
C106	191.32	191.32	191.32	191.32	0	0
R201	463.34	463.34	463.34	463.34	0	0
R202	410.55	410.55	410.55	410.55	0	0
R201	800.7	800.7	800.7	800.7	0	0
R202	712.25	712.25	712.25	712.25	0	0

TABLA 5.9: Resultados generados por *AS-GA*.

Instancia	Mejor reportado	Mejor	Peor	Promedio	σ^2	σ
C101	828.94	828.94	836.66	829.19	1.98	1.41
C105	828.94	828.94	830.6	828.99	0.09	0.30
C106	828.94	828.94	830.6	829.10	0.25	0.50
C107	828.94	828.94	841.41	829.46	5.27	2.29
C108	828.94	828.94	830.6	829.32	0.51	0.71
C109	828.94	828.94	830.6	828.04	0	0
C105	191.32	191.32	191.32	191.32	0	0
C106	191.32	191.32	191.32	191.32	0	0
R201	463.34	463.34	463.34	463.34	0	0
R202	410.55	410.55	410.55	410.55	0	0
R201	800.7	800.7	800.7	800.7	0	0
R202	712.25	712.25	712.25	712.25	0	0

TABLA 5.10: Resultados generados por *DSM-AS-PDA*.

2.2.- Remuestreo de los datos

El remuestreo permite estimar la precisión de las muestras estadísticas mediante el uso de subconjuntos de datos o tomando datos de manera aleatoria sobre un conjunto de datos (*bootstrapping*) [45]. En las Tablas 5.11, 5.12, 5.13, 5.14, 5.15, 5.16, 5.17 se muestran los resultados del remuestreo utilizando *bootstrap* para la media, desviación y varianza de los datos obtenidos por las técnicas en las instancias de prueba con un intervalo de confianza del 95 % utilizando 1000 muestras.

Instancia	Intervalo Medias		Intervalo Varianza		Intervalo Desviación	
	Lím. inf.	**Lím. sup.**	**Lím. inf.**	**Lím. sup.**	**Lím. inf.**	**Lím. sup.**
C101 (100)	848.39	934.65	5484.37	21660.14	74	147.17
C105 (100)	832.46	837.65	32.93	66.15	5.73	8.13
C106 (100)	830.21	834.07	15.16	51.68	3.89	7.189093
C107 (100)	831.96	836.68	31.5	53.18	5.61	7.29
C108 (100)	831.83	836.9	26.04	61.99	5.1	7.87
C109 (100)	829.66	831.46	0.6	21.86	0.77	4.67
C105 (50)	191.32	191.32	0	0	0	0
C106 (50)	191.32	191.32	0	0	0	0
R201 (50)	463.34	463.34	0	0	0	0
R202(50)	410.55	410.55	0	0	0	0
R201 (25)	800.73	800.73	0	0	0	0
R202 (25)	712.24	712.24	0	0	0	0

TABLA 5.11: Remuestreo *bootstrap* para *AS*.

Instancia	Intervalo Medias		Intervalo Varianza		Intervalo Desviación	
	Lím. inf.	**Lím. sup.**	**Lím. inf.**	**Lím. sup.**	**Lím. inf.**	**Lím. sup.**
C101 (100)	828.94	877.58	0	12241.89	0	110.64
C105 (100)	829.45	832.05	3.839769	27.71	1.95	5.26
C106 (100)	829.54	830.21	0.51	0.72	0.71	0.84
C107 (100)	831.84	836.42	28.82	40.28	5.36	6.34
C108 (100)	829.77	830.32	0.4	0.71	0.63	0.84
C109 (100)	829.71	830.27	0.46	0.72	0.67	0.84
C105 (50)	191.32	191.32	0	0	0	0
C106 (50)	191.32	191.32	0	0	0	0
R201(50)	463.34	463.34	0	0	0	0
R202(50)	410.55	410.55	0	0	0	0
R201(25)	800.73	800.73	0	0	0	0
R202(25)	712.24	712.24	0	0	0	0

TABLA 5.12: Remuestreo *bootstrap* para *AS-ILS*.

Instancia	Intervalo Medias		Intervalo Varianza		Intervalo Desviación	
	Lím. inf.	**Lím. sup.**	**Lím. inf.**	**Lím. sup.**	**Lím. inf.**	**Lím. sup.**
C101 (100)	828.93	859.45	0	3183.63	0	56.42
C105 (100)	829.3	837.5	1.27	508.77	1.13	22.55
C106 (100)	828.93	848.43	0	1338.25	0	36.58
C107 (100)	828.93	859.87	0	3625.77	0	60.21
C108 (100)	828.99	840.28	0.092	1111.6	0.3	33.34
C109 (100)	829.04	829.79	0.34	2.08	0.58	1.44
C105 (50)	191.32	191.32	0	0	0	0
C106 (50)	191.32	191.32	0	0	0	0
R201(50)	463.34	463.34	0	0	0	0
R202(50)	410.55	410.55	0	0	0	0
R201(25)	800.73	800.73	0	0	0	0
R202(25)	712.24	712.24	0	0	0	0

TABLA 5.13: Remuestreo *bootstrap* para *GA*.

Instancia	Intervalo Medias		Intervalo Varianza		Intervalo Desviación	
	Lím. inf.	**Lím. sup.**	**Lím. inf.**	**Lím. sup.**	**Lím. inf.**	**Lím. sup.**
C101 (100)	828.93	828.93	0	0	0	0
C105 (100)	828.93	830.22	0	8.57	0	2.92
C106 (100)	829.04	829.54	0.17	0.66	0.42	0.81
C107 (100)	828.93	831.84	0	28.82	0	5.36
C108 (100)	828.93	829.21	0	0.4	0	0.63
C109 (100)	828.93	828.93	0	0	0	0
C105 (50)	191.32	191.32	0	0	0	0
C106 (50)	191.32	191.32	0	0	0	0
R201(50)	463.34	463.34	0	0	0	0
R202(50)	410.55	410.55	0	0	0	0
R201(25)	800.73	800.73	0	0	0	0
R202(25)	712.24	712.24	0	0	0	0

TABLA 5.14: Remuestreo *bootstrap* para *AS-GA*.

Instancia	Intervalo Medias		Intervalo Varianza		Intervalo Desviación	
	Lím. inf.	**Lím. sup.**	**Lím. inf.**	**Lím. sup.**	**Lím. inf.**	**Lím. sup.**
C101 (100)	877.11	959.8	8879.04	19459.35	94.22	139.49
C105 (100)	834.14	837.05	9.74	23.64	3.12	4.86
C106 (100)	830.52	833.13	7.48	20.5	2.73	4.52
C107 (100)	833.55	837.65	24.93	37.73	4.99	6.14
C108 (100)	830.83	833.22	7.94	13.9	2.81	3.72
C109 (100)	829.77	830.32	0.4	0.71	0.63	0.84
C105 (50)	191.32	191.32	0	0	0	0
C106 (50)	191.32	191.32	0	0	0	0
R201(50)	463.34	463.34	0	0	0	0
R202(50)	410.55	410.55	0	0	0	0
R201(25)	800.73	800.73	0	0	0	0
R202(25)	712.24	712.24	0	0	0	0

TABLA 5.15: Remuestreo *bootstrap* para *HS*.

Instancia	Intervalo Medias		Intervalo Varianza		Intervalo Desviación	
	Lím. inf.	**Lím. sup.**	**Lím. inf.**	**Lím. sup.**	**Lím. inf.**	**Lím. sup.**
C101 (100)	828.93	866.84	0	10185.25	0	100.92
C105 (100)	829.96	832.54	7.13	15.35	2.67	3.91
C106 (100)	829.25	831.1	2.05	11.92	1.43	3.45
C107 (100)	829.61	832.87	6.95	31.72	2.63	5.63
C108 (100)	829.3	831.16	2.1	11.61	1.45	3.4
C109 (100)	829.15	829.71	0.33	0.71	0.57	0.84
C105 (50)	191.32	191.32	0	0	0	0
C106 (50)	191.32	191.32	0	0	0	0
R201(50)	463.34	463.34	0	0	0	0
R202(50)	410.55	410.55	0	0	0	0
R201(25)	800.73	800.73	0	0	0	0
R202(25)	712.24	712.24	0	0	0	0

TABLA 5.16: Remuestreo *bootstrap* para *AS-HS*.

Instancia	Intervalo Medias		Intervalo Varianza		Intervalo Desviación	
	Lím. inf.	Lím. sup.	Lím. inf.	Lím. sup.	Lím. inf.	Lím. sup.
C101 (100)	828.93	829.7	0	5.55	0	2.35
C105 (100)	828.93	829.1	0	0.25	0	0.5
C106 (100)	828.93	829.32	0	0.51	0	0.71
C107 (100)	828.93	830.4	0	14.45	0	3.8
C108 (100)	829.04	829.6	0.17	0.69	0.42	0.83
C109 (100)	828.93	829.21	0	0.4	0	0.63
C105 (50)	191.32	191.32	0	0	0	0
C106 (50)	191.32	191.32	0	0	0	0
R201(50)	463.34	463.34	0	0	0	0
R202(50)	410.55	410.55	0	0	0	0
R201(25)	800.73	800.73	0	0	0	0
R202(25)	712.24	712.24	0	0	0	0

TABLA 5.17: Remuestreo *bootstrap* para *DSM-AS-PDA*.

Con base en los datos mostrados en las Tablas 5.11, 5.12, 5.13, 5.14, 5.15, 5.16 y 5.17 se puede notar que *AS-GA* y *DSM-AS-PDA* son las técnicas que se comportan de una mejor forma, debido a que presentan un promedio cercano al mejor resultado reportado en la literatura para cada una de las instancias; sin embargo, *AS-ILS* y *AS-HS* se comportan mejor que las técnicas sencillas *AS*, *GA* y *HS*.

2.3.- Normalización de resultados

Con el objeto de comparar el comportamiento de las técnicas presentadas, los resultados obtenidos en las 12 instancias se normalizaron a través de la ecuación (5.1):

$$f(x^{norm-\alpha}) = \begin{cases} \frac{f(x^{worst\ in\ \beta}) - f(x^{method-\alpha}))}{f(x^{worst\ in\ \beta}) - f(x^{\star})} & \text{si } f(x^{worst\ in\ \beta}) - f(x^{\star}) \neq 0 \\ f(x^{worst\ in\ \beta}) - f(x^{method-\alpha}) & \text{si } f(x^{worst\ in\ \beta}) - f(x^{\star}) = 0 \\ 1 & \text{si } f(x^{worst\ in\ \beta}) = f(x^{\star}) \end{cases} \tag{5.1}$$

Donde, $f(x^{\star})$ es el mejor valor de la función objetivo encontrado; $f(x^{method-\alpha})$ es el promedio del valor de la función objetivo encontrado por la metaheurística α; $f(x^{worst\ in\ \beta})$ es el peor valor encontrado por las metaheurísticas para la instancia β y $f(x^{norm-\alpha})$ es el valor normalizado de la función objetivo encontrado por la metaheurística α. El valor de $f(x^{norm-\alpha})$ oscila en el rango 0 a 1. Si $f(x^{norm-\alpha})$ es cercano a 0, entonces el valor de$f(x^{method-\alpha})$ cercano al peor valor reportado. En contraste, si $f(x^{norm-\alpha})$ es cercano a 1, entonces el valor de $f(x^{method-\alpha})$ es cercano a $f(x^{\star})$. En la Tabla 5.18 se muestran los resultados normalizados para las técnicas implementadas en las 9 instancias de prueba.

	Normalización f(xnorm-θ)											
Téc.	**C101**	**C105**	**C106**	**C107**	**C108**	**C109**	**C105**	**C106**	**R201**	**R202**	**R201**	**R202**
AS	0.3	0.1	0.66	0.66	0	0	1	1	1	1	1	1
GA	0.84	0.54	0	0	0.27	0.7	1	1	1	1	1	1
HS	0	0	0.69	0.56	0.43	0.22	1	1	1	1	1	1
ASLS	0.77	0.79	0.92	0.7	0.81	0.26	1	1	1	1	1	1
ASHS	0.84	0.71	0.89	0.92	0.79	0.65	1	1	1	1	1	1
DSM-AS-PDA	1	1	1	1	1	1	1	1	1	1	1	1
ASGA	1	1	1	1	1	1	1	1	1	1	1	1

TABLA 5.18: Resultados normalizados.

En la Tabla 5.18, se puede notar que las técnicas *AS-GA* y *DSM-AS-PDA* obtienen los mejores resultados para las instancias de prueba. La ventaja principal de *DSM-AS-PDA* es el número de llamadas a la función objetivo, sin embargo, al depender de las soluciones que entrega el optimizador *Gurobi* la primera vez que se ejecuta la técnica para cierta instancia, el tiempo de cómputo se eleva. Por otro lado, una vez que se ha ejecutado *Gurobi* el tiempo de cómputo implicado es menor en comparación con las demás técnicas desarrolladas.

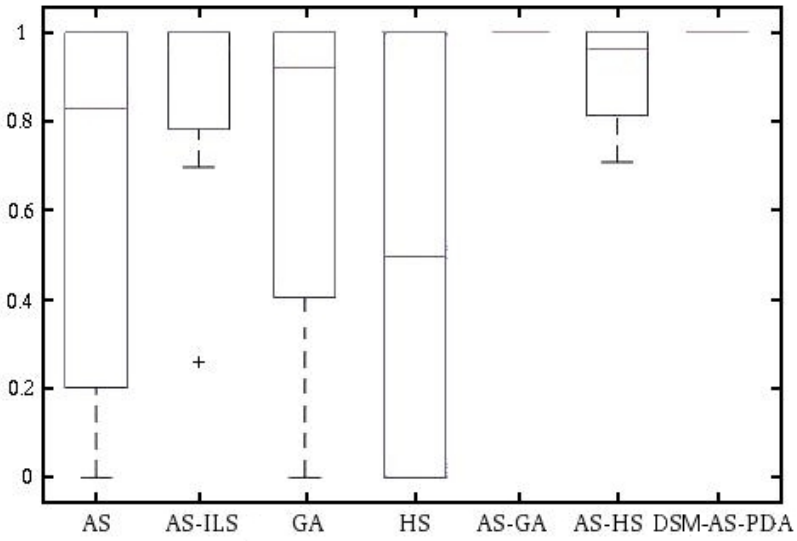

FIGURA 5.2: Diagrama de caja y bigote de los resultados normalizados.

En la Figura 5.2 se muestra gráficamente los resultados obtenidos por cada una de las técnicas. Aquí, se puede constatar que *AS-GA* y *DSM-AS-PDA* obtienen los mejores resultados, demostrando un comportamiento más robusto en comparación con las demás técnicas. Además, se puede notar que *AS-HS* y *AS-ILS* poseen un buen comportamiento obteniendo buenos resultados.

3.-Resultados comparativos

Para realizar un análisis sobre la robustez y el desempeño de las técnicas mostradas, se compararon los mejores resultados obtenidos por estas con los reportados en [1, 7, 83], que son de las últimas investigaciones para el problema.

En [7] los autores reportan una búsqueda en vecindades variables con operadores compuestos identificado como *(VNS-C)*, mientras que en [1] reportan un algoritmo de colonia artificial de abejas modificado identificado como *(ABC-M)*, y en [83] reportan tres heurísticas de búsqueda local denominadas como *Cuckoo search, Central Force Optimization, and Chemical Reaction Optimization*. En [7] reportan alrededor de 1,000,000 de evaluaciones a la función objetivo, en [1] reportan alrededor de 20,000 y en [83] reportan alrededor de 10,000.

Para poder realizar la comparación de los resultados obtenidos por las técnicas desarrolladas con las técnicas descritas en la literatura, se realizaron 30 ejecuciones de *AS-GA*, *AS-HS* Y *DSM-AS-PDA* para 9 instancias de prueba del conjunto propuesto por Cordeau y Solomon [17, 18, 84] de 100 clientes cada una (C101-C109).

En las Tablas 5.19 y 5.20 se muestran las comparaciones de los resultados obtenidos por *AS-GA, AS-HS Y DSM-AS-PDA* y los obtenidos en [1, 7, 83]; además, del número de llamadas a la función objetivo y los tiempos de ejecución (en segundos).

Para este trabajo se realizaron las ejecuciones en un equipo de cómputo con las siguientes características:

1. **Procesador:** Intel Core i5-4430, cuatro núcleos a 3.0 GHz.
2. **Memoria RAM:** 4 gb.

Inst.	**Resultados comparativos**								
	AS-HS			**AS-GA**			**DSM-AS-PDA**		
	Mejor	**Prom.**	**T(s).**	**Mejor**	**Prom.**	**T.(s)**	**Mejor**	**Prom.**	T(s).
C101	828.94	828.94	63.4	828.94	828.94	65.3	828.94	828.94	56.5
C102	828.94	830.84	57.8	828.94	829.45	53.6	828.94	829.67	49.6
C103	828.06	830.84	54.3	828.06	828.06	57.8	828.06	828.86	52.9
C104	824.78	825.95	56.7	824.78	824.96	52.4	824.78	824.78	53.4
C105	828.94	837.30	58.6	828.94	828.94	57.2	828.94	828.94	54.7
C106	828.94	829.43	56.3	828.94	828.94	53.7	828.94	828.94	55.8
C107	828.94	830.43	67.2	828.94	828.94	62.6	828.94	829.30	63.9
C108	828.94	828.94	76.2	828.94	828.94	73.6	828.94	828.94	64.5
C109	828.94	836.40	78.2	828.94	828.94	80.4	828.94	829.24	68.3

TABLA 5.19: Resultados comparativos (1).

Inst.	Resultados comparativos								
	Heurísticas			ABC-M			VNS-C		
	Mejor	Prom.	T. (s)	Mejor	Prom.	T. (s)	Mejor	Prom.	T. (s)
C101	828.94	–	–	828.94	828.94	700	828.94	828.94	–
C102	828.94	–	–	828.94	828.94	700	828.94	876.79	–
C103	828.06	–	–	828.94	840.66	700	828.94	832.65	–
C104	824.78	–	–	858.90	889.10	700	825.65	831.79	–
C105	828.94	–	–	828.94	828.94	700	828.94	852.33	–
C106	828.94	–	–	828.94	828.94	700	828.94	836.25	–
C107	828.94	–	–	828.94	828.94	700	828.94	853.9	–
C108	828.94	–	–	828.94	830.85	700	828.94	840.48	–
C109	828.94	–	–	828.94	836.97	700	828.94	828.94	–

TABLA 5.20: Resultados comparativos (2).

En las Tablas 5.19 y 5.20 se puede notar que las técnicas reportadas en la literatura y las descritas en este trabajo obtienen resultados similares. Además, se puede ver que los tiempos de ejecución para las técnicas *AS-GA, AS-HS y DSM-AS-PDA* son menores respecto a las técnicas mostradas en la literatura; sin embargo, estos no son buena medida de comparación, ya que para las diversas técnicas se utilizaron equipos de cómputo distintos para realizar las ejecuciones.

Por otro lado, se puede notar que el número de llamadas a la función objetivo también es menor en las técnicas desarrolladas y a su vez, da una mejor idea sobre el desempeño de las mismas.

Para comprobar que existen diferencias entre las distribuciones de las técnicas descritas en este trabajo con las reportadas en la literatura, se realizó la prueba de Wilcoxon. La hipótesis nula es que las muestras proceden de poblaciones con la misma distribución de probabilidad; la hipótesis alternativa establece que existe diferencia respecto a la tendencia central de las poblaciones.

En las Tablas 5.21, 5.22, 5.23, 5.24, 5.25 y 5.26 se muestran los valores de p y h obtenidos al aplicar la prueba para los mejores resultados reportados por cada técnica, el valor promedio de las ejecuciones y el tiempo de ejecución para cada una.

Técnica	Prueba Wilcoxon de los mejores valores para cada técnica (valor p)					
	AS-HS	AS-GA	DSM-AS-PDA	Heurísticas	ABC-M	VNS-C
AS-HS	1	1	1	1	0.25	0.25
AS-GA	1	1	1	1	0.25	0.25
DSM-AS-PDA	1	1	1	1	0.25	0.25
Heurísticas	1	1	1	1	0.25	0.25
ABC-M	0.25	0.25	0.25	0.25	1	0.5
VNS-C	0.25	0.25	0.25	0.25	0.5	1

TABLA 5.21: Prueba Wilcoxon de los mejores valores para cada técnica (valor p).

Tecnica	Prueba Wilcoxon de los mejores valores para cada técnica (valor h)					
	AS-HS	AS-GA	DSM-AS-PDA	Heurísticas	ABC-M	VNS-C
AS-HS	0	0	0	0	0	0
AS-GA	0	0	0	0	0	0
DSM-AS-PDA	0	0	0	0	0	0
Heurísticas	0	0	0	0	0	0
ABC-M	0	0	0	0	0	0
VNS-C	0	0	0	0	0	0

TABLA 5.22: Prueba Wilcoxon de los mejores valores para cada técnica (valor h).

Con base en los resultados mostrados en las Tablas 5.21 y 5.22 se muestra que todas las técnicas obtienen resultados estadísticamente similares, esto es debido a que en su mayoría se entrega como mejor valor encontrado al mejor resultado descrito en la literatura.

Técnica	Prueba Wilcoxon para los resultados promedio (valor p)					
	AS-HS	AS-GA	DSM-AS-PDA	Heurísticas	ABC-M	VNS-C
AS-HS	1	0.015625	0.015625	-	0.64	0.0546875
AS-GA	0.015625	1	0.125	-	0.125	0.015625
DSM-AS-PDA	0.015625	0.125	1	-	0.15625	0.015625
Heurísticas	-	-	-	-	-	
ABC-M	0.64	0.125	0.15625	-	1	0.546875
VNS-C	0.0546875	0.015625	0.015625	-	0.546875	1

TABLA 5.23: Prueba Wilcoxon para los resultados promedio (valor p).

Técnica	Prueba Wilcoxon para los resultados promedio (valor h)					
	AS-HS	AS-GA	DSM-AS-PDA	Heurísticas	ABC-M	VNS-C
AS-HS	0	1	1	-	0	0
AS-GA	1	0	0	-	0	1
DSM-AS-PDA	1	0	0	-	0	1
Heurísticas	-	-	-	-	-	-
ABC-M	0	0	0	-	0	0
VNS-C	0	1	1	-	0	0

TABLA 5.24: Prueba Wilcoxon para los resultados promedio (valor h).

En las Tablas 5.23 y 5.24 se observa que las técnicas *AS-GA*, *DSM-AS-PDA* y *ABC-M* obtienen en promedio resultados estadísticamente similares; sin embargo, con la información mostrada en las Tablas 5.19 y 5.20 se observa que la principal ventaja de *AS-GA* y *DSM-AS-PDA* es el número de llamadas a la función objetivo que estas realizan en comparación a *ABC-M*.

Técnica	Prueba Wilcoxon para tiempos de ejecución (valor p)					
	AS-HS	**AS-GA**	**DSM-AS-PDA**	**Heurísticas**	**ABC-M**	**VNS-C**
AS-HS	1	0.1953125	0.00390625	-	0.00390625	-
AS-GA	0.1953125	1	0.0546875	-	0.00390625	-
DSM-AS-PDA	0.00390625	0.0546875	1	-	0.00390625	-
Heurísticas	-	-	-	-	-	
ABC-M	0.00390625	0.00390625	0.00390625	-	1	-
VNS-C	**-**	**-**	**-**	**-**	**-**	**-**

TABLA 5.25: Prueba Wilcoxon para tiempos de ejecución (valor p).

Técnica	Prueba Wilcoxon para tiempos de ejecución (valor h)					
	AS-HS	**AS-GA**	**DSM-AS-PDA**	**Heurísticas**	**ABC-M**	**VNS-C**
AS-HS	0	0	1	-	1	-
AS-GA	0	0	0	-	1	-
DSM-AS-PDA	1	0	0	-	1	-
Heurísticas	-	-	-	-	-	-
ABC-M	1	1	1	-	0	-
VNS-C	-	-	-	-	-	-

TABLA 5.26: Prueba Wilcoxon para tiempos de ejecución (valor h).

En las Tablas 5.25 y 5.26 se observa que *AS-HS*, *AS-GA* y *DSM-AS-PDA* tienen tiempos de ejecución estadísticamente similares, mientras que *VNS-C* tiene un tiempo de ejecución distinto a las demás. Esto se debe principalmente, al número de llamadas a la función objetivo, ya que *VNS-C* realiza un número considerablemente mayor de llamadas.

4.- Conclusiones y trabajo futuro

En este trabajo, se presentaron 7 técnicas para resolver el *VRP-TW*, de las cuales se hace énfasis en los algoritmos híbridos basados en técnicas heurísticas y métodos de programación matemática. Es destacable que todas las técnicas tienen un buen desempeño al compararlas con los resultados reportados en la literatura especializada; sin embargo, los algoritmos híbridos *AS-GA* y *DSM-AS-PDA* son capaces de generar buenos resultados empleando menos recursos computacionales respecto a las demás técnicas.

Además, éstos son capaces de generar mejores resultados que los métodos reportados en la literatura para algunas instancias pertenecientes al conjunto de casos de prueba, que fue propuesto por Cordeau *et al.* Se puede observar que los algoritmos híbridos emplean menos llamadas a la función objetivo en contraste con los métodos básicos *(AS, GA, HS, DSM, PDA*), ya que en cada uno, la estructura del problema se utilizó para guiar la búsqueda de una mejor forma.

Por otro lado, los métodos *AS-GA* y *DSM-AS-PDA* son capaces de generar buenos resultados con menos del 25 % de las llamadas a la función objetivo respecto a la técnica mostrada en [83], que es de las técnicas que reportan pocas llamadas a la función objetivo. A su vez, al realizar la prueba de Wilcoxon, se demuestra que se obtienen buenos resultados utilizando menos recursos computacionales y obteniendo mejores promedios que las técnicas reportadas en la literatura.

Como trabajo futuro, se se propone realizar versiones autoadaptativas y/o reactivas para cada algoritmo híbrido presentado, con el fin de obtener un comportamiento más inteligente y reducir más las llamadas a la función objetivo; además, se busca atacar diversas versiones de *VRP* e instancias de problemáticas reales.

Se cumplió el objetivo del presente trabajo, gracias a que se obtuvieron técnicas que son competitivas ante las técnicas reportadas en la literatura específica.

Apéndice A

Principales funciones de la interfaz *Gurobi C*

A.1.- Solución de un modelo

Una vez que haya construido el modelo, se puede llamar a *GRBoptimize* para calcular una solución. Por defecto, *GRBoptimize()* utilizará optimización simultanea (resuelve el programa lineal con distintas estrategias) para resolver modelos de PL, el algoritmo de barrera para resolver QP y modelos QCP, y el algoritmo de ramificación y corte para resolver modelos enteros mixtos. La solución se almacena como un conjunto de atributos del modelo.

Los algoritmos de *Gurobi* dan un seguimiento cuidadoso al estado del modelo, por lo que la llamada *GRBoptimize()* sólo realizará una optimización adicional si los datos relevantes han cambiado desde la última vez que el modelo fue optimizado[1]. Por otro lado, se puede descartar información de soluciones previamente calculadas y se reinicia la optimización sin necesidad de cambiar el modelo con la rutina *GRBresetmodel*.

Si se tiene que un modelo resulta infactible, se cuenta con algunas opciones para tratar de diagnosticar la causa de la infactibilidad y repararla. Para esto, se llama a la rutina *GRBcomputeIIS*. También se puede llamar a *GRBfeasrelax* para calcular una relajación de factibilidad para el modelo. Esta relajación permite encontrar una solución que minimiza la magnitud de la violación de las restricciones.

A.1.1.- Modificación de atributos

La mayor parte de la información asociada con un modelo dentro de *Gurobi* se almacena en un conjunto de atributos. Algunos atributos están asociados con las variables del modelo, algunos con las restricciones del modelo, y algunas de ellas con el propio modelo. Para dar un ejemplo sencillo, la solución

[1]Después de que un modelo MIP ha sido resuelto, puede llamar *GRBfixedmodel* para calcular el modelo fijo asociado. Este modelo es idéntico al modelo de entrada, excepto que todas las variables enteras se fijan a sus valores en la solución MIP.

de un modelo hace que los atributos de la variable **X** que se calculan por el optimizador Gurobi no pueden ser modificados directamente por el usuario.

La interfaz *Gurobi C* contiene un amplio conjunto de rutinas para consultar o modificar los valores de los atributos. La rutina exacta a utilizar para un atributo particular depende del tipo del atributo. Como se mencionó anteriormente, los atributos pueden ser tanto los atributos de variable, atributos de restricción o atributos del modelo.

Los atributos de variables y restricciones son matrices, y se debe utilizar el conjunto de rutinas de atributos matriz. Los atributos del modelo son escalares, y se debe utilizar el conjunto de rutinas escalares. Y algunos valores de atributos pueden ser de tipo *char , int , doble o de cadena*.

Los atributos de modelo escalares se acceden a través de un conjunto de rutinas *GRBget*attr()*, por ejemplo, *GRBgetintattr*. Además, esos atributos se pueden establecer directamente por el usuario. Los atributos de tipo matriz se acceden a través de tres conjuntos de rutinas. El primer conjunto, las *GRBget*attrarray()* devuelven una sub-serie de la matriz de atributos, especifica el uso del índice del primer elemento y la longitud de la sub-serie deseada.

El segundo conjunto, las rutinas *GRBget*attrelement()* devuelven una sola entrada del conjunto de atributos. Por último, las *GRBget*attrlist()* recuperan los valores de atributos para obtener una lista de índices. Los atributos de la matriz que se pueden configurar por el usuario se modifican a través de los *GRBset*attrarray()*, *GRBset*attrelement()* y *GRBset*attrlist()*.

La matriz de restricciones puede ser modificada de distintas maneras. La primera es llamar a *GRBchgcoeffs* cambiando los coeficientes de la matriz. Esta rutina se puede utilizar para modificar el valor de una variable existente que no sea cero.

La matriz de restricciones también se modifica cuando se quitan las restricciones (a través de *GRBdelconstrs*) o variables (a través *GRBdelvars*). Los valores no nulos asociados a las restricciones o variables eliminadas se quitan junto con las restricciones propias o variables.

Los términos objetivos cuadráticos se añaden a la función objetivo utilizando la rutina *GRBaddqpterms*. Se puede añadir una lista de términos cuadráticos en una llamada, o se pueden agregar términos de forma incremental a través de múltiples llamadas. La rutina *GRBdelq* permite borrar todos los términos de segundo grado a partir del modelo. Se debe tener en cuenta que los modelos cuadráticos suelen tener ambos términos cuadráticos y lineales.

A.1.2.- Gestión de los parámetros

Gurobi proporciona un conjunto de parámetros que le permite controlar muchos de los detalles del proceso de optimización. Factores como tolerancias de factibilidad y de optimalidad, opciones de algoritmos, las estrategias

para explorar el árbol de búsqueda, etc., los cuales se pueden controlar mediante la modificación de parámetros antes de comenzar la optimización. Los parámetros se ajustan usando las rutinas *GRBset*param()*. Los valores actuales se pueden recuperar con *GRBgetdblparam*.

Los parámetros pueden ser de tipo *int , doble , o char**. También se puede leer un conjunto de valores de los parámetros desde un archivo usando *GRBreadparams*, o escribir el conjunto de parámetros modificados utilizando *GRBwriteparams*.

También, se incluye una herramienta de ajuste de parámetros automatizado que explora muchos conjuntos diferentes de cambios de parámetros con el fin de encontrar un conjunto que mejore el rendimiento. Una cosa que debemos tener en cuenta es que cada modelo tiene su propia copia del entorno cuando se crea. Los cambios de parámetros en el entorno original no tienen ningún efecto en los modelos existentes.

A.1.3.- Seguimiento del Progreso

De forma predeterminada, *Gurobi* envía controles simples que están disponibles para modificar el comportamiento del registro. Se puede modificar la salida a un archivo, especificando el nombre de archivo en *GRBloadenv* cuando se crea su entorno. Mientras que, la rutina *GRBsetcallbackfunc* permite instalar una función que *Gurobi* llamará regularmente durante el proceso de optimización. Se puede llamar *GRBcbget* desde el interior de la devolución de llamada para obtener información adicional sobre el estado de la optimización.

Estas se pueden utilizar para modificar el comportamiento de *Gurobi*. Si se llama a la rutina *GRBterminate* desde el interior de una devolución de llamada, por ejemplo, el optimizador termina en el punto conveniente más temprano. **La rutina *GRBcbsolution* permite inyectar una solución factible (o solución parcial) durante la solución de un modelo**. Las rutinas *GRBcbcut y GRBcblazy* permiten añadir planos de corte y restricciones relajadas durante el proceso de optimización.

A.2.- Rutinas principales en Gurobi

A.2.1.- GRBloadmodel

Sirve para crear un nuevo modelo de optimización, utilizando los argumentos proporcionados para inicializar los datos del mismo (función objetivo, matriz de restricciones, etc.). Si la matriz de restricciones contiene más de 2 mil millones de valores distintos de cero, se debe considerar el uso de la *GRBXloadmodel* variante de esta rutina. Si la función regresa un valor distinto de cero indica que se produjo un problema al crear el modelo.

- **Argumentos**

  1. **env:** El entorno en el que se debe crear el nuevo modelo. Se debe de tener en cuenta que el nuevo modelo crea una copia de este entorno, por lo que las modificaciones posteriores en el entorno original no afectarán el nuevo modelo.
  2. **modelP:** La ubicación en la que se debe colocar el apuntador para el nuevo moelo.
  3. **Pname:** El nombre del modelo.
  4. **numvars:** El número de variables en el modelo.
  5. **numconstrs:** El número de restricciones en el modelo.
  6. **objsense:** El sentido de la función objetivo. Los valores permitidos son 1 (minimización) o -1 (maximización).
  7. **objcon:** Desplazamiento constante del objetivo.
  8. **obj:** Coeficientes objetivo para las nuevas variables. Este argumento puede ser nulo, en cuyo caso los coeficientes objetivo se establecen en 0.
  9. **sense:** Los sentidos de las nuevas restricciones. Las opciones son " $=$ " (igual), $'<'$ (menor que o igual), o $'>'$ (mayor que o igual). También se pueden usar constantes GRB, GRB_LESS_EQUAL, o $GRB\_GREATER\_EQUAL$.
  10. **dcha:** Los valores del lado derecho de las nuevas restricciones. Este argumento puede ser nulo, en cuyo caso los valores del lado derecho se establecen en 0.
  11. **vbeg:** Los valores de la matriz de restricciones que no son cero son pasados a esta rutina. Cada columna de la matriz de restricciones se representa como una lista de índice de valor par y cada entrada proporciona el valor correspondiente. Cada variable en el modelo tiene un valor **vbeg** y **vlen**, indicando la posición de inicio de los no ceros para esa variable en los arreglos **vind** y **vval**, y el número de valores distintos de cero para esa variable, respectivamente. Así, por ejemplo, si $vbeg[2] = 10$ y $vlen[2] = 2$, indicaría que la variable 2 tiene dos valores distintos de cero asociados. Sus índices de restricción se pueden encontrar en $vind[10]$ y $vind[11]$, y los valores numéricos para los no-ceros se pueden encontrar en $vval[10]$ y $vval[11]$.
  12. **vlen:** Es el número de valores distintos de cero en las restricciones asociadas con cada variable.
  13. **ind:** Índices de restricción asociados con valores no nulos.
  14. **vval:** Los valores numéricos asociados con la matriz de restricción distintos de cero.
  15. **lb:** Límites inferiores para las nuevas variables. Este argumento puede ser nulo, en cuyo caso todas las variables consiguen límites inferiores de 0.

16. **ub:** Límites superiores para las nuevas variables. Este argumento puede ser nulo, en cuyo caso todas las variables obtienen cotas superiores infinitas.
17. **vtype:** Tipos de las variables. Las opciones son $GRB\_CONTINUOUS$, GRB_BINARY, $GRB\_INTEGER$, $GRB\_SEMICONT$, o $GRB\_SEMIINT$. Este argumento puede ser $NULL$, en cuyo caso todas las variables se supone que son continuas.
18. **varnames:** Los nombres de las nuevas variables. Este argumento puede ser nulo, en cuyo caso todas las variables se les asignan nombres predeterminados.
19. **constrnames:** Los nombres de las nuevas restricciones. Este argumento puede ser nulo, en cuyo caso todas las restricciones se dan nombres predeterminados.

Es recomendable que se construya un modelo de una restricción/variable a la vez, usando *GRBaddconstr* o *GRBaddvar*, en lugar de utilizar esta rutina para cargar toda la matriz de restricciones a la vez.

- **Ejemplo de uso**

GRBloadmodel(env, &model, "example", vars, constrs, -1, 0.0, obj, sense, rhs, vbeg, vlen, vind, vval, NULL, NULL, vtype, NULL, NULL);

A.2.2.- GRBnewmodel

Crea un nuevo modelo de optimización. Esta rutina permite especificar un conjunto inicial de variables (con los límites, los tipos y nombres), pero el modelo inicial no tendrá restricciones. Las restricciones pueden ser añadidos más tarde con *GRBaddconstr* o *GRBaddconstrs*.

- **Argumentos:** Utiliza los argumentos **env, modelIP, Pname, numvars, obj, lb ub, vtipe, varnames** descritos anteriormente.
- **Ejemplo de uso**

GRBnewmodel(env, &model, "New", 2, obj, NULL, NULL, NULL, names);

A.2.3.- GRBaddvars

Añade nuevas variables a un modelo existente. Se debe tener en cuenta que las nuevas variables no se añadirán en realidad hasta la siguiente llamada a *GRBoptimize* o *GRBupdatemodel*.

- **Argumentos:**

  model: El modelo al que hay que añadir las nuevas variables. **numnz:** El número total de coeficientes distintos de cero en las nuevas columnas. Además de **numvarz, vbeg, vind, vval, obj, lb, ub, vtypw, varnames** descritos anteriormente.

- **Ejemplo de uso**

GRBaddvars(*model,numvars,numnz,*vbeg,*vind,*vval,*obj,*lb,*ub,*vtype,**varnames)

A.2.4.- GRBaddconstr

Añade una nueva restricción a un modelo existente. Se debe tener en cuenta que la nueva restricción no será añadida en realidad hasta la siguiente llamada a *GRBoptimize* o *GRBupdatemodel*.

- **Argumentos:**

  cind: índices de variables para valores distintos de cero en la nueva restricción. **cval:** Los valores numéricos de los valores distintos de cero en la nueva restricción. Además de **model, numnz, sense, dcha y constrname** descritas anteriormente.

- **Ejemplo de uso**

GRBaddconstr(model, 3, ind, val, GRB_EQUAL, 1.0, “New”);

A.2.5.- GRBoptimize

Optimiza un modelo. El algoritmo utilizado para la optimización depende del tipo de modelo.

- **Argumentos:**

  model descrito anteriormente.

- **Ejemplo de uso**

GRBoptimize(model);

A.2.6.- GRBcomputeIIS

Calcula un subsistema reducido inconsistente *(IIS)*. Un *IIS* es un subconjunto de las restricciones y límites variables del modelo original. Si se eliminan

todas las restricciones en el modelo, excepto los de *IIS*, el modelo sigue siendo infactible. Sin embargo, la eliminación de cualquier miembro de *IIS* produce un resultado factible.

- **Argumentos:**

  model descrito anteriormente.

- **Ejemplo de uso**

GRBcomputeIIS(model);

A.2.7.- GRBfeasrelax

Modifica el modelo de entrada para crear una relajación de factibilidad. Se debe tener en cuenta que es necesario llamar a *GRBoptimize* para calcular la solución del problema relajado.

La relajación de factibilidad es un modelo que, cuando se resuelve, reduce al mínimo la cantidad en que la solución viola los límites y restricciones lineales del modelo original.

Si se especifica $relaxobjtype = 0$, el objetivo de la relajación de factibilidad es minimizar la suma de las magnitudes ponderadas de las violaciones y de restricciones. Si se especifica $relaxobjtype = 1$, el objetivo es minimizar la suma ponderada de los cuadrados de las violaciones y de restricciones. Si se especifica $relaxobjtype = 2$, el objetivo es minimizar el conteo ponderado de violaciones consolidados y de restricciones [2].

- **Argumentos:**

  1. **relaxobjtype:** La función de costos aplicado al encontrar la relajación de costo mínimo.
  2. **minrelax:** El tipo de relajación de factibilidad a realizar.
  3. **lbpen:** La penalización asociada con la violación de un límite inferior.
  4. **ubpen:** La penalización asociada con la violación de un límite superior.
  5. **rhspen:** La penalización asociada a violar una restricción lineal.
  6. **feasobjP:** Cuando $minrelax = 1$, esto devuelve el valor objetivo para la relajación de costo mínimo

  Además de **model** descrito anteriormente.

- **Ejemplo de uso**

[2]Se debe tener en cuenta que esta es una rutina destructiva, ya que modifica el modelo que se le ha pasado.

double penalties[]; error = GRBfeasrelax(model, 0, 0, NULL, NULL, penalties, NULL); error = GRBoptimize(model);

Apéndice B

Modelos matemáticos del *VRP-TW* utilizados en *Gurobi*

Para ejemplificar las relajaciones aplicadas al modelo original, se utilizó una instancia de 3 clientes y 2 vehículos tomando la información de los primeros 3 clientes de la instancia *C101* del conjunto propuesto por Cordeau y Solomon [19, 84]. En la Tabla B.1, se muestra la información para esta instancia.

Cliente	Coord. x	Coord. y	T. servicio	Demanda	Límite inicial de tiempo	Límite final de tiempo
0	40.00	50.00	0.00	0.00	1236	–
1	45.00	68.00	90.00	10.00	912	967
2	45.00	70.00	90.00	30.00	825	870
3	42.00	66.00	90.00	10.00	65	146

TABLA B.1: Instancia de muestra.

B.1.- Relajación respecto a capacidad vehícular e integralidad

Aquí, se muestra la formulación matemática del *VRP-TW* eliminando las restricciones de capacidad de los vehículos e integralidad de las variables, este modelo está formulado siguiendo el formato *lp* de *Gurobi*.

Minimize C
Subject To
X010001 + X020001 + X030001 + X010002 + X020002 + X030002 = 1
X000101 + X020101 + X030101 + X000102 + X020102 + X030102 = 1
X000201 + X010201 + X030201 + X000202 + X010202 + X030202 = 1
X000301 + X010301 + X020301 + X000302 + X010302 + X020302 = 1
X010001 - X000101 + X020001 - X000201 + X030001 - X000301 = 0
X010002 - X000102 + X020002 - X000202 + X030002 - X000302 = 0
X000101 - X010001 + X020101 - X010201 + X030101 - X010301 = 0
X000102 - X010002 + X020102 - X010202 + X030102 - X010302 = 0
X000201 - X020001 + X010201 - X020101 + X030201 - X020301 = 0
X000202 - X020002 + X010202 - X020102 + X030202 - X020302 = 0

X000301 - X030001 + X010301 - X030101 + X020301 - X030201 = 0
X000302 - X030002 + X010302 - X030102 + X020302 - X030202 = 0
b0001 <= 1236
b0001 >= 1
b0101 >= 912
b0101 <= 967
b0201 >= 825
b0201 <= 870
b0301 >= 65
b0301 <= 146
b0002 <= 1236
b0002 >= 1
b0102 >= 912
b0102 <= 967
b0202 >= 825
b0202 <= 870
b0302 >= 65
b0302 <= 146
C - 43.8634 X000101 - 20.6155 X000201 - 16.1245 X000301 - 43.8634 X010001 - 45.0444 X010201 - 42.0476 X010301 - 20.6155 X020001 - 45.0444 X020101 - 5 X020301 - 16.1245 X030001 - 42.0476 X030101 - 5 X030201 - 43.8634 X000102 - 20.6155 X000202 - 16.1245 X000302 - 43.8634 X010002 - 45.0444 X010202 - 42.0476 X010302 - 20.6155 X020002 - 45.0444 X020102 - 5 X020302 - 16.1245 X030002 - 42.0476 X030102 - 5 X030202 >= 0
generals
b0001
b0101
b0201
b0301
b0002
b0102
b0202
b0302
binary
X000101
X000201
X000301
X010001
X010201
X010301
X020001
X020101
X020301
X030001
X030101
X030201
X000102

X000202
X000302
X010002
X010202
X010302
X020002
X020102
X020302
X030002
X030102
X030202
End

B.2.- Relajación respecto a ventanas de tiempo e integralidad

Aquí, se muestra la formulación matemática del *VRP-TW* eliminando las restricciones de tiempo para cada cliente, además de la integralidad de las variables, este modelo está formulado siguiendo el formato *lp* de *Gurobi*.

Minimize C
Subject To
X010001 + X020001 + X030001 + X010002 + X020002 + X030002 = 1
X000101 + X020101 + X030101 + X000102 + X020102 + X030102 = 1
X000201 + X010201 + X030201 + X000202 + X010202 + X030202 = 1
X000301 + X010301 + X020301 + X000302 + X010302 + X020302 = 1
X010001 - X000101 + X020001 - X000201 + X030001 - X000301 = 0
X010002 - X000102 + X020002 - X000202 + X030002 - X000302 = 0
X000101 - X010001 + X020101 - X010201 + X030101 - X010301 = 0
X000102 - X010002 + X020102 - X010202 + X030102 - X010302 = 0
X000201 - X020001 + X010201 - X020101 + X030201 - X020301 = 0
X000202 - X020002 + X010202 - X020102 + X030202 - X020302 = 0
X000301 - X030001 + X010301 - X030101 + X020301 - X030201 = 0
X000302 - X030002 + X010302 - X030102 + X020302 - X030202 = 0
Y0001 - 0 X000101 - 0 X000201 - 0 X000301 = 0
Y0002 - 0 X000102 - 0 X000202 - 0 X000302 = 0
Y0101 - 10 X010001 - 10 X010201 - 10 X010301 = 0
Y0102 - 10 X010002 - 10 X010202 - 10 X010302 = 0
Y0201 - 30 X020001 - 30 X020101 - 30 X020301 = 0
Y0202 - 30 X020002 - 30 X020102 - 30 X020302 = 0
Y0301 - 10 X030001 - 10 X030101 - 10 X030201 = 0
Y0302 - 10 X030002 - 10 X030102 - 10 X030202 = 0
Y0001 + Y0101 + Y0201 + Y0301 <= 0
Y0002 + Y0102 + Y0202 + Y0302 <= 0

```
Y0001 >= 0
Y0002 >= 0
Y0101 >= 0
Y0102 >= 0
Y0201 >= 0
Y0202 >= 0
Y0301 >= 0
Y0302 >= 0
C - 43.8634 X000101 - 20.6155 X000201 - 16.1245 X000301 - 43.8634 X010001 -
45.0444 X010201 - 42.0476 X010301 - 20.6155 X020001 - 45.0444 X020101 - 5
X020301 - 16.1245 X030001 - 42.0476 X030101 - 5 X030201 - 43.8634 X000102 -
20.6155 X000202 - 16.1245 X000302 - 43.8634 X010002 - 45.0444 X010202 - 42.0476
X010302 - 20.6155 X020002 - 45.0444 X020102 - 5 X020302 - 16.1245 X030002 -
42.0476 X030102 - 5 X030202 >= 0
binary
X000101
X000201
X010001
X010201
X010301
X020001
X020101
X020301
X030001
X030101
X030201
X000102
X000202
X010002
X010202
X010302
X020002
X020102
X020302
X030002
X030102
X030202
generals
Y0101
Y0102
Y0201
Y0202
Y0301
Y0302
End
```

Bibliografía

[1] M. Alzaqebah, S. Abdullah y S. Jawarneh. «Modified artificial bee colony for the vehicle routing problems with time Windows.» En: *Research (Springer)*, 1 (2016.), págs. 1-10.

[2] N. Azi, M. Gendreau y J. Y. Potvin. «An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicle». En: *European Journal of Operational Research.* (2010).

[3] M.L. Balinski y R.E. Quandt. «On an Integer Program for a Delivery Problem.» En: *Operations Research, 12 (2).* (1965).

[4] J.E. Beasley. «Route-first cluster-second methods for vehicle routing.» En: *Omega 11.* (1983).

[5] J. Berger, M. Barkaoui y O.A. Bräysy. «Route-Directed Hybrid Genetic Approach for the Vehicle Routing Problem with Time Windows.» En: *INFOR, Vol. 41, No. 2.* (2003).

[6] J. Berger, Salois M. y R.A. Begin. «Hybrid Genetic Algorithm for the Vehicle Routing Problem with Time Windows». En: *Lecture Notes in Artificial In telligence 1418, Springer, Berlin.* (1998).

[7] C. Binhui y col. «A Variable Neighbourhood Search Algorithm with Compound Neighbourhoods for VRPTW.» En: *The 2016 International Conference on Operations Research and Enterprise Systems (ICORES'16), 23-25.* (2016.).

[8] J.L. Blanton y R.L. Wainwright. «Multiple Vehicle Routing with Time and Capacity Constraints Using Genetic Algorithms.» En: *Proceedings of the 5th International Conference on Genetic Algorithms, San Francisco.* (1993).

[9] W.E. Bolkan. «Algorithm for the Vehicle Routing Problem with Deadlines.» Tesis de lic. Department of Computer Science y Operations Research, North Dakota State University, Fargo, North Dakota., 1986.

[10] A. Bouthillier y T.G. Crainic. «A Cooperative Parallel Meta-Heuristic for the Vehicle Routing Problem with Time Windows.» En: *Computers & Operations Research, Vol. 32, No. 7.* (2005).

[11] J. Bramel y D. Simchi-Levi. «A location based heuristic for general routing problems.» En: *Operations Research 43.* (1995).

[12] X. Chen, W.S. Wan y X. H. Xu. «The Real-Time Time-Dependent Vehicle Routing Problem.» En: *Transportation Research Part E: Logistics and Tran sportation Review, Vol. 42, No. 5.* (2006).

[13] Y.-J. Cho y S.-D. Wang. «A Threshold Accepting MetaHeuristic for the Vehicle Routing Prob lem with Backhauls and Time Windows.» En: *Journal of the E astern Asia Society for Transportation Studies, Vol. 6.* (2005).

[14] N. Christofides, A. Mingozzi y P. Toth. «The Vehicle Routing Problem.» En: *Combinatorial Optimization.* (1979).

[15] G. Clarke y Wright J.V. «Scheduling of vehicles from a central depot to a number of delivery points.» En: *Operations Research vol.12 pp.568-581.* (1964).

[16] C.A. Coello Coello. *Introducción a la Computación Evolutiva.* Ed. por CINVESTAV-IPN. CINVESTAV-IPN., 2014.

[17] J. F. Cordeau y G. T. Laporte. «The dial-a-ride problem: models and algorithms.» En: *Annals of Operations Research.* (2007).

[18] J. F. Cordeau y col. «Vehicle Routing.» En: *Transportation, handbooks in operations research and management science.* (2006).

[19] J.F. Cordeau y col. «The VRP with time windows.» En: *Technical Report Cahiers du GERAD G-99-13.* (1999).

[20] R.J. Dakin. «A Tree-Search Algorithm for Mixed Integer Programming Problems. Computer Journal, Vol. 8, pp 250-255, 1964.» En: *Computer Journal, Vol. 8, pp 250-255.* (1964).

[21] G.B. Dantzig y J.H. Ramser. «The truck dispatching problem.» En: *Management Science vol. 6.* (1959).

[22] G.B. Dantzig y P. Wolfe. «Decomposition Principle for Linear Programs.» En: *Operations Research 8(1), 101-111.* (1960).

[23] G.B. Dantzig y P. Wolfe. «The Decomposition Algorithm for Linear Programs.» En: *Econometrica 29(4), 767-778.* (1961).

[24] M. Desrochers y T.W. Verhoog. «A matching based savings algorithm for the vehicle routing problem.» En: *Cahier du GERAD G-89-04. École des Hautes Études Commerciales de Montréal.* (1989).

[25] M. Dorigo. «Optimization, Learning and Natural Algorithms.» Tesis doct. Dipartimento di Elettronica, Politecnico di Milano, Milan., 1992.

[26] M. Dorigo y L. M. Gambardella. «Ant Colony System: A cooperative learning approach to the traveling salesman problem.» En: *IEEE Transactions on Evolutionary Computation.* (1997).

[27] M. Dorigo, V. Maniezzo y A. Colorni. «Ant System: Optimization by a Colony of Cooperating Agents.» En: *IEEE Transactions on Systems, Man and Cybernetics 26 (1).* (1996).

[28] J. Dréo y col. *Metaheuristics for Hard Optimization, Methods and Cases Studies'*. Ed. por Springer. Springer., 2006.

[29] G. Dueck y T. Scheuer. «Threshold accepting: A general purpose optimization algori thm appearing superior to simulated annealing.» En: *Journal of Computation Physics, 90 :161-175.* (1990).

[30] I. Dumitrescu y T. Stuetzle. «Combinations of local search and exact algorithms.» En: *In G. R. Raidl et al., editors, Applications of Evolutionary Computation, volume 2611 of LNCS, pages 211–223. Springer.* (2003).

[31] W.-G. Eilon y N. Christofides. «Distribution Management.» En: *Mathematical Modelling and Practical.* (1971).

[32] J.w. Escobar, R. Linfati y W. Adarme-Jaimes. «A hybrid metaheuristic algorithm for the capacitated location routing problem.» En: *Revista DYNA, Facultad de Minas, Medellín Colombia.* (2014).

[33] M.L. Fisher. «Comments of The Lagrangian Relaxation Method for Solving Integer Programming Problems.» En: *Management Science 50(12), 1872-1874.* (2004).

[34] M.L. Fisher. «The Lagrangian Relaxation Method for Solving Integer Programming Problems.» En: *Management Science 50(12), 1861-1871.* (2004).

[35] M.L. Fisher y R. Jaikumar. «A Decomposition Algorithm for Large-Scale Vehicle Routing.» En: *Decision Sciences Working Paper 78-11-05, University of Pennsylvania.* (1978).

[36] M.L. Fisher y R. Jaikumar. «A Generalized Assignment Heuristic for Vehicle Routing.» En: *Networks, Vol. 11.* (1981).

[37] M. Flood. «The Traveling-Salesman Problem». En: *Operational Research, vol. 4* (1956).

[38] A. García-Najera y J.A. Bullinaria. «An Improved Multi-Objective Evolutionary Algorithm for the Vehicle Routing Problem with Time Windows.» En: *Computers & Operations Research, Vol. 38, No. 1.* (2011).

[39] T.J. Gaskell. «Bases for vehicle fleet scheduling.» En: *Operations Research Quarterly, 18, 281–95.* (1967).

[40] J. Kim Geem Z. y G.V. Loganathan. «A New Heuristic Optimization Algorithm: Harmony Search.» En: *Simulation.* (2001).

[41] M. Gendreau, A. Hertz y G. Laporte. «A new insertion and postoptimization procedures for the traveling salesman problem.» En: *Operations Research, 40(6).* (1992).

[42] F. Glover. «Future paths for integer programming and links to artificial intellingence.» En: *Computers & Operations Research, 5 : 533-549.* (1986).

[43] M. X. Goemans y 144-191. ISO 690 Williamson D. P.). «The primal-dual method for approximation algorithms and its application to network design problems.» En: *Approximation algorithms for NP-hard problems.* (1997).

[44] R.E. Gomory. «Outline of an algorithm for integer solutions to linear programs.» En: *Bull. Am. Math.* (1958).

[45] P. Good. *Introduction to Statistics Through Resampling Methods and R/S-PLUS.* Wiley., 2005.

[46] *Gurobi Optimization Manual Reference, 2016. "http://www.gurobi.com"*.

[47] K.H. Hansen y J. Krarup. «Improvements of the Held -Karp Algorithm for the Symmetric Travelling Salesman Problem.» En: *Mathematical Programming, 7 : 87-96.* (1974).

[48] M. Held y Karp. R.M. «The traveling salesman problem and minimum spanning trees.» En: *Operations Research 18, 1138-1162.* (1970).

[49] M. Held y Karp. R.M. «The traveling salesman problem and minimum spanning trees: Part II.» En: *Mathematical Programming 1, 6-25.* (1971).

[50] W.K. Ho, J.C. Agn y A. Lim. «A Hybrid Search Algorithm for the Vehicle Routing Problem with Time Windows.» En: *International Journal on Artificial Intelligence Tools.* (2001).

[51] J. H. Holland. «Outline for a logical theory of adaptive systems.» En: *Journal of the Association for Computing Machinery.* (1962).

[52] J. Homberger y H. Gehring. «A Two-Phase Hybrid Meta-Heuristic for the Vehicle Routing Problem with Time Windows.» En: *European Journal of Operational Research, Vol. 162, No. 1.* (2005).

[53] J.J. Hopfield y D. Tank. «Neural computation of decisions in optimization problems.» En: *Biological Cybernetics, 52 : 141-152.* (1985).

[54] Potvin J.Y. y S. Bengio. «The Vehicle Routing Problem with Time Windows Part II: Genetic Search.» En: *Informs Journal on Computing, Vol. 8.* (1996).

[55] N. Khanh, T.G. Crainic y M. Toulouse. «A hybrid generational genetic algorithm for the periodic vehicle routing problem with time windows.» En: *Journal of Heuristics, Volume 20, Issue 4, pp 383-416.* (2014).

[56] S. Kirkpatrichk, C.D. Gellat y M.P. Vecchi. «Optimization by simulated annealing.» En: *Science, 220(4598): 671-680.* (1983).

[57] Y.A. Kokosidis, W.B. Powell y M.M. Solomon. «An Optimization-Based Heuristic for Vehicle Routing and Scheduling with Soft Time Window Constraints.» En: *Transportation Science, 26(2) :69-85.* (1992).

[58] G. Laporte. «The Vehicle Routing Problem: An overview of exact and approximate algorithms.» En: *European Journal of Operational Research, Vol. 59.* (1991).

[59] G. Laporte, Mercure H. e Y. Nobert. «An exact algorithm for the asymmetrical capacitated vehicle routing problem.» En: *Networks 16.* (1986).

[60] K.S. Lee y Z.W. Geem. «new metaheuristic algorithm for continuous engineering optimization, harmony search theory and practice.» En: *A Computer Methods in Applied Mechanics Engineering,* (2005).

[61] J. K. Lenstra y A. H. G. Rinnooy Kan. «Complexity of vehicle routing and scheduling problems.» En: *Networks.* (1981).

[62] S. Lin y B.W. Kernigham. «An effective heuristic algorithm for the traveling salesman problem.» En: *Operations Research, 21.* (1973).

[63] J. D. C. Little y col. «An Algorithm for the Traveling Salesman Problem.» En: *Operations Research, Vol. 11, pp 979- 989.* (1968).

[64] M. Mahdavi, M. Fesanghary y E. Damangir. «An improved harmony search algorithm for solving optimization problems.» En: *Applied Mathematics Computation.* (2007).

[65] V. Maniezzo, T. Stützle y S. Vo. *Matheuristics. Hybridizing Metaheuristics and Mathematical Programming, Volume 10.* Springer., 2014.

[66] O. Martin, H.R. Lourenço y T. Stützle. «A beginner's introduction to iterated local search.» En: *In Metaheuristics International Conference, MIC 2001.* 2001.

[67] O. Martin, H. R. Lurenço y T. Stützle. *Handbook of Metaheuristics, Iterated Local Search.* Kluwer Academic, 2003.

[68] C. Miller, A. Tucker y R. Zemlin. «Integer programming formulations and traveling salesman problems.» En: *J. of the ACM.* (1960).

[69] R.H. Mole y S.R. Jameson. «A sequential route-building algorithm employing a generalised savings criterion.» En: *Operations Research Quarterly.* (1976).

[70] Y. Nagata, O. Braysy y W. Dullaert. «A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows.» En: *Computers and Operations Research.* (2010).

[71] A. Olivera. «Heurísticas para Problemas de Ruteo de Vehículos.» En: *Universidad de la República, Montevideo, Uruguay.* (2004).

[72] I. Or. «Traveling Salesman-Type Combinatorial Problems and their relation to the Logistics of Blood Banking.» Tesis doct. Ph.D. Thesis, Department of Industrial Engineering y Management Science, Northwestern University, Evanston, IL.., 1976.

[73] J.P. Orrego-Cardozo. *Solución al Problema de Ruteo de Vehículos con Capacidad Limitada CVRP a través de la heurística de barrido y la implementación del Algortmo Genético de Chu-Beasley.* Inf. téc. Universidad Tecnológica de Pereira, Facultad de Ingeniería Industrial., 2013.

[74] C. H. Papadimitriou y K. Steiglitz. *Combinatorial Optimization Algorithms and Complexity,* Dover Publications, 1998.

[75] K.-W. Peng. «An Adaptive Parallel Route Construction Heuristic for the Vehicle Routing Problem with Time Windows Constraints.» En: *Expert Systems with Applications, Vol. 38, No. 9.* (2011).

[76] K.V. Price, R.M. Storn y J.A. Lampinen. «Differential Evolution A Practical Approach to Global Optimization. Natural Computing Series.» En: *Springer-Verlag, Berlin, Germany.* (2005).

[77] J. Puchinger y G. R. Raidl. «Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification.» En: *In J. Mira and J. Alvarez, editors, Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation, volume 3562 of LNCS, pages 41-53. Springer.* (2005).

[78] H. Pullen y M. A. Webb. «Computer application to a transport scheduling problem.» En: *Computer Journal, 10, pp. 10–13.* (1967).

[79] M. Reimann, s. Shtovba y C E. Nepomuceno. «A hybrid ACO-GA approach to solve Vehicle Routing Problems.» En: (2001).

[80] P.P. Repoussis, C.D. Tarantilis y G. Ioannou. «Arc-guided evolutionary algorithm for the vehicle routing problem with time windows.» En: *IEEE Transactions on Evolutionary Computation, v.13.* (2009).

[81] Y. Rochat y F. Semet. «A tabu search approach for delivering pet food and flour in Switzerland.» En: *Journal of the Operational Research Society, 45.* (1994).

[82] D.M. Ryan, C Hjorring y F. Glover. «Extension of the petal method for vehicle routing». En: *J. Opl Res, Soc. 44.* (1993).

[83] T. Saeheaw y N. Charoenchai. «Comparison of Meta-heuristic Algorithms for Vehicle Routing Problem with Time Windows.» En: *Advanced Computer and Communication Engineering Technology.* (2015.).

[84] M. Solomon. «Algorithms for Vehicle Routing and Scheduling Problems with time window constraints.» En: *Northeastern University, Boston, Massachusetts.* (1985).

[85] M. Solomon y J. Desrosiers. «Time Window constrained routing scheduling Problems: A Survey.» En: *Forthcoming in Transportation Science.* (1988).

[86] K. Soonpracha, A. Mungwattana y T. Manisri. «Re-constructed Meta-Heuristic Algorithm for Robust Fleet Size and Mix Vehicle Routing Problem with Time Windows under Uncertain Demands.» En: *Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems - Volume 2.* (2015).

[87] R. Storn y K. Price. «Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces.» En: *J. of Global Optimization 11(4).* (1997).

[88] E.J. Swenson. «The Vehicle Routing Problem with Time Constraints.» Tesis de lic. Department of Computer Science y Operations Research, North Dakota State University, Fargo, North Dakota., 1986.

[89] Ma. Tai-Yu. «A cross entropy multiagent learning algorithm for solving vehicle routing problems with time windows.» En: *Proceedings of the 2nd International Conference on Computational Logistics.* (2011).

[90] S.R. Thangiah, Osman I.H. y T. Sun. «Hybrid Genetic Algorithms, Simulated Annealing and Tabu Search Methods for Vehicle Routing Problems with Time Windows.» En: *Technical Report UKC/OR94/4, Institute of Mathematics and Statistics, University of Kent, Canterbury.* (1995).

[91] P.M. Thompson y H.N. Psaraftis. «Cyclic transfer algorithms for multivehicle routing and scheduling problems.» En: *Operations Research, 41 (5): 935-946.* (1993).

[92] P. Toth y D. Vigo. «An Overview of Vehicle Routing Problems. Monographs on Discrete Mathematics and Applications. In: The Vehicle Routing Problem.» En: *SIAM.* (2000).

[93] A. Van Breedam. «Improvement heuristics for the vehicle routing problem based on simulated annealing.» En: *European Journal of Operational Research, 86 : 480-490.* (1995).

[94] S. Villagra y col. «Metaheurísticas híbridas y paralelas aplicadas a problemas de ruteo de vehículos.» En: *WICC 2014 XVI Workshop de Investigadores en Ciencias de la Computación.* (2014).

[95] A. Wren y Carr J.D. «Computers in transport planning and operation.» En: (1971).

[96] P. Yellow. «A computational modification to the savings method of vehicle scheduling.» En: *Operations Research Quarterly, 21, 281–83.* (1970).

[97] Y. Zhou y col. «A Hybrid Bat Algorithm with Path Relinking for the Capacitated Vehicle Routing Problem.» En: *Mathematical Problems in Engineering.* (2013).